

IN THE SPECIFICATION:

Replace the previous version of the specification with the following substitute specification. Another version of the specification, marked up to show all changes relative to the previous version of the specification is included in Appendix A.

Express Mail Label No. EL 653 505 325 US

OIPE JC165
MAY 0 3 2001
PATENT & TRADEMARK OFFICE

RECEIVED
MAY 7 - 2001
Technology Center 2100

A Method and System for Discovery of Trades Between Parties

1 Related Applications

This application claims priority to provisional application no. 60/168,754 filed on December 6, 1999, titled, "An E-Commerce Infrastructure for Value Chains", the contents of which are herein incorporated by reference. This application also claims priority to provisional application no. 60/194,880, titled, "Method and System to Mediate Commerce", filed on April 6, 2000, the contents of which are herein incorporated by reference.

09425350 042701

2 Field of the Invention

The invention relates to a method and system for discovery of trades between parties. In particular, the invention is a system which allows buyers to define their preferences and sellers to define their capabilities, then determines which trading points maximize the utility of the buyer. The system suggests trades by exploiting the flexibilities and tradeoffs encoded by both parties, thus providing win-win trades. A second level of optimization ranks the trades with all suppliers, allowing the buyer to rapidly determine the best alternatives. The system allows for rich negotiation spaces and supports continuous, discrete, and range or interval decision factors.

3 Background of the Invention

The present invention relates to methods of automatic exploration and exploitation of the flexibilities possessed by negotiating parties to uncover improved win-win agreements. The invention describes computationally efficient mechanisms that are applicable whether there are one or many selling parties. The precise number and types of negotiating dimensions are irrelevant as long as they are numerical. Thus the present invention applies equally to the optimal determination of terms in the purchase of a commodity or an arbitrarily complex artifact.

The past 5-10 years have seen remarkable growth in software tools to help firms with enterprise-wide planning (ERP software) and supply chain management (SCM software). While these tools do a wonderful job at integrating disparate data sources within and between firms, the opportunity exists for significant further cost reductions.

The same time period has also seen a tremendous rise in the widespread use of the internet by both consumers and businesses. Forecasters are predicting that within a few years e-commerce between businesses (B2B) and between consumers and businesses (B2C) will grow to in excess of a trillion dollars per year in annual revenues.

Electronic markets have proliferated over the last few years with the advent of B2C (business-to-consumer) and B2B (business-to-business) electronic commerce. Such market places have yielded significant cost savings by lowering the transaction costs between buyers and sellers. Buyers have also profited through increased competition between suppliers. However, electronic markets have hurt suppliers, since the zero-sum negotiation over price has been at their expense. The present invention describes a tool whereby cost savings for both parties are derived from the discovery of win-win trades. Fundamentally, the system works by allowing trading parties to describe their desired trade across multiple dimensions and to express their flexibility around this ideal trade. Through an algorithmic exploration of their flexibilities, the present invention can discover trades that are near the ideal trades of both parties, enabling both to win.

The adoption of B2B and B2C electronic commerce was facilitated by the migration of catalogues online. This familiar method of presentation ameliorated the significant cultural change to electronic trade. For the foreseeable future, electronic commerce will be dominated by online catalogs. At present, online catalogues are direct translations of their hardcopy counterparts where the attributes of a product are described and a price quoted. Inevitably however, online catalogs will become more expressive. Catalog entries will be able to represent price breaks for large quantity orders, lot sizes, etc. Thus it is important that any software (like the present invention) that uncovers mutually beneficial trading scenarios is able to operate with such catalogs. Consequently, in the present invention there is an asymmetry between buyer (usually a human) and seller (usually an online catalog).

One of the reasons catalogs have come to dominate electronic commerce is that the types of goods that can be represented in catalogs are simple. Whether the product is pens or paper clips, different vendor's offers differ little from each other (a pen is a pen is a pen), and a quick scan of a catalog gives a buyer enough information to make an informed purchase. These types of goods are low margin and inexpensive. In contrast, the vast amount of purchasing between businesses involves materials which are directly connected with business operations - car parts, turbines, etc. Such direct goods are the future of electronic commerce. Unlike present-day engines, any truly useful procurement tool must be able to support direct materials with complex attributes and complex inter-relationships between its components.

Electronic commerce offers unprecedented opportunities for more informed decision-making for both buyers and sellers. The past few decades have seen the widespread adoption of enterprise resource planning (ERP) systems, to the point that now almost every major company has some form of ERP software. ERP functions as the digital nervous system of a company, transmitting and logging information between the company's many different business functions. ERP software keeps track of inventory, monitors the state of purchase orders, signals when a company should reorder direct and indirect materials, and a myriad of other functions.

Consequently, ERP databases are a rich source of information to optimize a company's operations. Yet today this information is rarely used to make more informed buying and selling decisions. The present invention can utilize such information sources to optimize a company's interactions with suppliers and customers.

One important manner in which this optimization can occur is through an analysis of *all* cost factors. Current buying and selling practices often focus on limited goals, e.g., minimize the total purchase price. Myopic purchasing strategies often result in higher total cost of ownership when all cost factors relevant to a product in its lifetime of use are included. These other cost factors can be significant. Why save the money in taking delivery two days late if the receiving docks will be full at that time and an additional shift needs to be hired to clear the docks? Why order the cheaper drill bit if it is much more expensive to replace when it breaks? The present invention improves trades by minimizing the total cost of ownership of a product, yielding significant savings to its users. Many total cost factors are difficult to quantify – e.g. what is the cost of dealing with a unionized versus a non-unionized supplier? Consequently, the present invention supports qualitative (best guesses and intuition) as well quantitative factors.

All companies are situated in a supply or value chain. At each step in the chain, a company purchases from its suppliers, transforms these inputs, and sells the output to its customers. The termination of the supply chain is the sale of the final product to the end consumer. Since the only influx of external capital comes from the end consumer, companies have realized that they compete not only as individuals but also as entire supply chains. As result, software products have recently become available which attempt to streamline the operations of links within the entire supply chain. This software, variously called supply chain optimization (SCO) or advanced planning and optimization (APO), operates on the basis of forecasted demand at various points within the supply chain. Based on these predictions, plans are generated telling companies how much to produce and how to schedule their operations. SCO systems are a valuable source of intra-company information – data the present invention capitalizes on. Because SCO software relies on forecasted demand, it is only as helpful as the forecast is accurate, and, unfortunately, in many cases demand is very difficult to predict. How can the software know that laundry detergent will go on special at grocery stores in the Northeast in 7 weeks? As a result of the volatility in demand and the many other unpredictable perturbations that plague supply chains, companies keep significant buffers in the form of inventories. In addition to planning, businesses must also be able to adapt to unplanned effects. Such adaptation requires flexibility and a means to exploit that flexibility. The present invention exploits the flexibility of trading parties to streamline the operations of supply chains by smoothing the boundaries between trading parties.

The present invention is therefore a system to allow trading parties to express trading desires and constraints across many and varied different factors. These trading preferences are informed by many different data sources to optimize for a company's internal operations and its connections to it's supply chain through an analysis including total cost factors. The flexibility expressed by all trading parties is exploited to locate win-win opportunities for all

parties if they exist.

4 Summary of the Invention

We describe the present invention in its application to facilitating trade between buyers and sellers, but note that the mechanisms described are much more general. We can easily imagine, for example, using the present invention to match individuals (with the desires and skills) to projects.

The inspiration for the present invention comes from utility theory developed by economists since the 1960's. Since we are interested in multiple dimensions of negotiation, we draw from the multi-attribute utility theory literature.[1] Utility is an abstract concept which has been formalized in various ways. For the present purposes utility, u, is a number between 0 and 1 representing a party's willingness to trade. Larger values indicate a greater willingness.

4.1 The Negotiation Space

In any negotiation the parties must come to agreement on the factors requiring negotiation. We call these factors dimensions or variables. As an example, when purchasing a car, the buyer may be concerned with price, time of delivery, and color. Each factor price, time, and color is a dimension. Most dimensions can be classed as one of three types: continuous, discrete, or range/interval. A continuous dimension is one like price for which the buyer's utility varies smoothly across that dimension. The buyer's utility at \$23 001.00 is almost the same as the utility at \$23 000. Color is a discrete dimension. Since the car may only be available in black, white, and silver, the domain of this dimension is the finite set of values {black, white, silver}. Moreover, the buyer's utility may be quite different for the three colors. The third class of dimensions is called interval dimensions. An interval dimension arises often in B2B negotiations. If a machined part is built to some tolerance (e.g., the inner diameter of a screw is between 24.5 and 25.5 mm), the range of variability in the dimension is specified as an interval. In the language of statistical quality control, a certain percentage of the machined parts will fall in this range. These three broad classes of variables capture almost all the types of attributes relevant to B2B negotiation.

The present invention operates over any number of continuous, discrete, and range or interval variables. We call the negotiation space X and any point in the negotiation space $(\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r}) \in X$. It is important to recognize that the single trading point $(\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r})$ may have multiple components, e.g., price = \$23 000, time of delivery = 3 weeks, color = black.

In the present invention, the space of negotiation is agreed upon by all parties involved prior to the commencement of any negotiation. We can, however, imagine more dynamic situations in which dimensions are introduced and discarded over time.

[1] For a good introduction to multi-attribute utility theory see [1].

4.2 The Buyer's Utility Function

A party defines it's utility function over this space so that every $(\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r})$ is assigned a utility number indicating the party's willingness to trade. We indicate the utility function as $u((\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r}))$. A great deal of work has been done on the appropriate form for utility functions. In the present invention, we take a simple form for the utility function for two reasons. First, we would like the form of the utility to be conducive to rapid computation. Second we would like the utility to be simple enough to be easily understood by and elicited from users of the invention. With no loss in generality, we write the utility function as $u((\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r})) = \exp(-d((\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r})))$ where $d(\mathbf{x})$ is interpreted as the distance of trading point $(\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r})$ from the most preferred trade.

So that we can operate against seller catalogs, only the buyer needs to define a utility function. Across the continuous dimensions, the buyer's utility is defined by specifying the most preferred (or ideal) continuous dimensions and the manner in which utility drops off as we move away from this ideal. For the discrete dimensions, the utility is specified in tabular form since there are a finite number of alternatives. Again, the buyer must specify it's ideal discrete values and how utility decays away from those values. In section 6.1 we describe how this is accomplished. The range dimensions contribute to utility similarly; the buyer specifies an ideal range and the utility decays for ranges other than the ideal according to their distance from the ideal.

The utility function can also express tradeoffs between variables, e.g., I may take delivery in 5 weeks if the price drops to $20 000, or I may accept the white car if I can take delivery in 2 weeks. The tradeoffs may be between pairs of continuous dimensions (as in the first case), between pairs of discrete variables, or between continuous and discrete variables (as in the second case).

4.2.1 Normalization and Weighting

When utility is defined over different types of variables, it is important to normalize the contributions of each variable so that the buyer can weight the importance of the various contributions to utility. This is a difficult problem. How should a buyer's color preferences be normalized so that they can be traded off against time of delivery? The present invention solves this problem by requiring that the average distance of any negotiation variable from its ideal value is the same for all dimensions. Since the buyer is more interested in those regions of the negotiation space where the utility is high, the average is weighted by utility. This procedure defines a manner in which to define a baseline where all dimensions contribute equally. Given this baseline, the buyer can then weight the various contributions and obtain useful results.

4.2.2 Utility Elicitation

Since utility is fundamental to the present invention, its elicitation from the buyer is important. Utility may be defined using any of a number of sources:

1. graphical user interfaces associated with the invention
2. standard benchmark criteria applicable to the domain
3. formal methodologies like the analytical hierarchical process [2], or discrete choice analysis [3]
4. inferred through models

We expand briefly upon method 4. As discussed in the background section, it is important to buyers to minimize their total cost of ownership. If we have a function representing these costs as a function of the negotiation variables, and perhaps other factors, this function can be used to infer a utility function which will act to minimize the total costs. Later we describe how this can be accomplished.

4.3 A Supplier's Capabilities

As noted previously, suppliers are treated differently from buyers so that the tool can operate against catalog information with no human intervention required on the part of the seller. In fact, we do not require sellers to define a utility at all.

A supplier cannot offer deals at all points within the negotiation space X, e.g., he certainly can't offer the black car tomorrow for free! A capability then represents the ability of a supplier to deliver and defines a subspace of X. It can include such things as price discounts on large volume orders, variation in delivery time as a function of price, etc. Since these relationships are already specified by businesses in terms of simple rules like *"the price per unit is $10.00 if 1 to 999 units are ordered and $9.50 per unit if 1000 or more units are ordered"*, suppliers' capabilities are represented in the present invention by piecewise linear functions.

4.4 Negotiation Constraints

Both parties may have constraints which must be satisfied in order for them to trade. For example, the buyer may not buy the car unless he gets it within 6 weeks, or he may not purchase the car if it is available only in white. These are examples of continuous and discrete constraints, respectively. A continuous constraint sets a requirement on the continuous variables. In the present invention, continuous constraints must be either linear or quadratic. Discrete constraints involve discrete variables. A discrete constraint can be expressed as a list of the allowed (or disallowed) combinations of the discrete variables for which the trade will be acceptable. For example, if the buyer would accept either the black or the silver car, the constraint would list both these colors as viable. It is important to note that both continuous and discrete constraints may involve one or more variables. We can also express constraints involving both types of variables by allowing the continuous constraints to differ depending on the discrete variables.

4.5 Utility Optimization

With the major components of the invention in place, we describe how the overall system works. As a procurement tool for the buyer, there are two levels of optimization. First, for any given supplier we maximize the buyer's utility, subject to the supplier's capabilities to find that trade which makes the buyer as happy as possible. Since we are optimizing within a supplier's capabilities, the supplier has expressed a willingness to complete the trade at whatever point is determined to be optimal. The tool then optimizes across suppliers to rank them according to utility at the optimal point. A graphical user interface allows a buyer to investigate the trades suggested by the tool by sorting according to any dimension or by the overall utility.

Utility, while a useful concept in assessing an overall score, may be of limited use to a buyer due to its abstract meaning. Consequently, we can also apply the total cost of ownership function to the results to rank order the suggested trades according to their various cost components. Recall that for any trade $\mathbf{x} \in X$, the total cost of ownership function returns the various cost contributions. This additional information aids the buyer in his purchasing decision. The utility number for each trade is still useful because the total cost of purchase function includes only those cost factors which can be quantified, whereas the utility also includes "softer" qualitative factors.

4.5.1 Aggregation

In addition to optimizing against one supplier at a time, the present invention can also be used to optimize against an arbitrary aggregation of suppliers. This is important if, for example, no single seller can supply the large volume requested by a buyer. In this mode of operation, the buyer specifies sets of suppliers participating in the aggregation and the dimensions over which aggregation can occur, and the tool finds the optimal combination in which to distribute the volume dimension over the allowed suppliers.

4.6 An e-Commerce Infrastructure for Value Chains

This patent application also describes an integrated solution for B2C and B2B e-commerce that would be built on top of ERP and SCM software and which would provide a number of compelling benefits to companies. Amongst the benefits are:

- multidimensional markets which allow consumers to implicitly define their preferences over many criteria. This allows both consumers to express what it is they really value, allows companies to position themselves clearly in the space of value, and allows for efficient matches between trading partners

- optional anonymity of market participants and their trading desires when that is appropriate[2]

[2]Anonymity is not, however, a pre-requisite of the proposed invention.

- explicit pricing of the flexibility possessed by the consumer and all businesses in the supply chain which allows for more robust operation of the entire supply chain. This concept is very different from other types of markets (*e.g.* auctions, reverse auctions, exchanges) where transactions are specified exactly. The flexibility introduced by any party, whether consumer or supplier, is propagated and exploited through the entire supply chain.

- capture and quantification of true consumer demand leading to improved forecasting and product development by suppliers

- automated markets that integrate supply chain networks through coordination across and within company boundaries. Coupling of the automated markets with local (*i.e.* at the company level) optimization tools fed by real-time company data allows for optimization and cost savings across the entire supply chain.

It should be recognized that supply chains may be very different in the near future. Current supply chains are based on physical objects made valuable through a sequence of transformations resulting in a product purchased by an end consumer. With the move to an information economy the supply chain of the near future may not involve physical goods at all. In particular the entire supply chain may consist of value adding operations converting raw data to consumer-desired information. Such supply chains will have the same coordination problems current ones do. Our proposed solution applies equally well to these future supply chains and by supply chain we mean this more general notion.

5 Brief Description of the Figures

Figure 1 shows an architecture for the invention.
Figure 2 shows a schematic of a buyer-specific capability with examples indicating potential input.
Figure 3 shows a schematic of a supplier-specific preference with examples indicating potential input.

6 Detailed Description

6.1 Theory

In this section we outline the mathematical foundations of the optimization process in sufficient detail to allow for computer implementation.

6.1.1 The Negotiation Space

In Table 1 we define the parameters which collectively define the space of negotiation X. For each of the n_c continuous variables, we specify an allowed range over which that continuous dimension may vary as $x_i \in X_i = [\underline{x}_i, \overline{x}_i]$, where $\underline{\mathbf{x}}$ is the n_c-vector of lower continuous bounds

n_c	number of continuous dimensions
n_d	number of discrete dimensions
n_r	number of range dimensions
$\mathbf{x}$	n_c-vector of values for continuous dimensions
$\boldsymbol{\varkappa}$	n_d-vector of values for continuous dimensions
x_i	value of ith continuous variable
$\varkappa_i$	value of ith discrete variable

Table 1: Definition of the negotiation search space.

and $\overline{\mathbf{x}}$ is the n_c-vector of upper continuous bounds. Each discrete variable assumes a value from within its domain $\varkappa_i \in \mathcal{D}_i$. Without loss of generality, we label the domain of discrete variable i by $\mathcal{D}_i = [1, \cdots, d_i]$ where $d_i \geq 0$ is an integer giving the number of possible values discrete variable $\varkappa_i$ may assume.

With these definitions, we define the space of negotiation by the tensor product $X = X_1 \otimes \cdots \otimes X_{n_c} \otimes \mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_{n_d}$. Range variables are treated separately and not negotiated over.

6.1.2 The Utility Function

The utility function is a mapping from X into the interval $[0, 1]$. As indicated earlier we assume the utility to have the form $u(\mathbf{x}, \boldsymbol{\varkappa}) = \exp[-d(\mathbf{x}, \boldsymbol{\varkappa})]$ where $d(\mathbf{x}, \boldsymbol{\varkappa})$ is interpreted as a distance. In what follows we will assume that in its simplest form the distance function has the form

$$d(\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r}) = \big(\mathbf{x} - \boldsymbol{\mu}(\boldsymbol{\varkappa})\big)^t \mathbf{C}^{-1}(\boldsymbol{\varkappa})\big(\mathbf{x} - \boldsymbol{\mu}(\boldsymbol{\varkappa})\big) + Z(\boldsymbol{\varkappa}) + R(\mathbf{r}; \boldsymbol{\varkappa}).$$

Each contribution to the distance function is positive. We consider each contribution to the distance in turn, beginning with the range variable contribution $R(\mathbf{r}; \boldsymbol{\varkappa})$.

First, we note that the range distance depends on the setting of the discrete variables. This allows the buyer to express different preferences for the range variables depending on discrete factors. The total range distance is summed up over all possible range variables so that $R(\mathbf{r}; \boldsymbol{\varkappa}) = \sum_{i=1}^{n_r} R_i(r_i; \boldsymbol{\varkappa})$. The vector $\mathbf{r}$ indicates the preferred values for all range variables. If range variable i is specified as the interval $r_i \equiv (\underline{r}_i, \overline{r}_i)$ (where $\overline{r}_i > \underline{r}_i$) then $\mathbf{r}$ is an n_r-vector of such tuples. The distance contribution, R_i, from one range variable will depend on the application. If the range variables are meant to represent the tolerances on machined parts where issues of statistical quality control are important, then the distance between two ranges might be related to the overlap between Gaussian distributions. If r_i is interpreted as a Gaussian having mean $(\underline{r}_i + \overline{r}_i)/2$ and standard deviation proportional to $\overline{r}_i - \underline{r}_i$ then an appropriate range distance is given in Appendix A. Other choices for the range distance function are certainly possible.

The continuous distance is quadratic and determined by the positive semidefinite $n_c \times n_c$ matrix $\mathbf{C}^{-1}$. We have allowed this matrix to vary with the setting of the discrete variables and indicated this explicitly through $\mathbf{C}^{-1}(\boldsymbol{\varkappa})$.[3] The n_c-vector $\boldsymbol{\mu}$ may also depend on $\boldsymbol{\varkappa}$ and indicates

[3]Where no confusion will arise we eliminate the dependence of $\mathbf{C}^{-1}$ on $\boldsymbol{\varkappa}$ for notational simplicity.

the point at which the utility is maximal – $\boldsymbol{\mu}$ is thus identified with the ideal value for the continuous variables. The precise quadratic form is convenient, but, using recent developments in interior point methods, other convex functions are also computationally tractable [4].

The discrete distance is determined through the function $Z(\boldsymbol{\varkappa})$ which maps the discrete space $\mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_{n_d}$ onto the positive real line $[0, \infty]$. In keeping with the assumption that distance is a function of only pairs of components x_i, x_j, we assume the discrete distance has the form[4]

$$Z(\boldsymbol{\varkappa}) = \sum_{i=1}^{n_d} \Big\{ Z_i(\varkappa_i) + \sum_{j=1(\neq i)}^{n_d} Z_{i,j}(\varkappa_i, \varkappa_j) \Big\}.$$

Each contribution $Z_{i,j}$ is a table consisting of $d_i d_j$ entries, where $Z_{i,j}(\varkappa_i, \varkappa_j)$ can be interpreted as the distance if discrete dimension i has value $\varkappa_i$ conditioned on discrete dimension j having value $\varkappa_j$. The diagonal terms Z_i offer an unconditional distance. The most preferred value for the ith discrete dimension is that for which $Z_i(\varkappa_i) = 0$.

Rather than require the user to enter the distances explicitly, there are numerous ways in which the distances can be generated automatically based upon a buyer's ranking of preferred values. Further details can be found in Appendix B.

Weighting of Dimensions

In many cases it is important for simple modifications of the distance function to re-weight the contributions to the total distance. If $\mathbf{w}_c$ is an n_c-vector of weights for the continuous dimensions, we can accomplish this by letting $\mathbf{C}_w^{-1} = \mathbf{W}_c \mathbf{C}^{-1} \mathbf{W}_c$ where $\mathbf{W}_c$ is the diagonal matrix $\mathbf{W}_c = \mathrm{diag}(\mathbf{w}_c)$.[5] In a similar way we modify the discrete distance to $Z_{w;i,j}(\varkappa_i, \varkappa_j) = w_{d;i} w_{d;j} Z_{i,j}(\varkappa_i, \varkappa_j)$ where $\mathbf{w}_d$ is the n_d-vector of weights for the discrete variables and $w_{d;i}$ is its ith component. The range contribution is also modified so that $R_{w;i}(r_i) = w_{r;i} R_i(r_i)$ where $\mathbf{w}_r$ is the n_r-vector of weights for the range variables and $w_{r;i}$ is its ith component. For convenience the weights are normalized so that $(\mathbf{1}^t \mathbf{w}_c)^2 + (\mathbf{1}^t \mathbf{w}_d)^2 + \mathbf{1}^t \mathbf{w}_r = 1$. With little additional complexity the dimension weights can be made dependent on the setting of the discrete variables but we will assume throughout that the weights are constant.

With these modifications, the total distance function becomes

$$d(\mathbf{x}, \boldsymbol{\varkappa}) = \big(\mathbf{x} - \boldsymbol{\mu}(\boldsymbol{\varkappa})\big)^t \mathbf{C}_w^{-1}(\boldsymbol{\varkappa}) \big(\mathbf{x} - \boldsymbol{\mu}(\boldsymbol{\varkappa})\big) + Z_w(\boldsymbol{\varkappa}) + R_w(\mathbf{r}). \tag{1}$$

where $Z_w(\boldsymbol{\varkappa}) = \sum_{i=1}^{n_d} w_{d;i} \big\{ w_{d;i} Z_i(\varkappa_i) + \sum_{j=1(\neq i)}^{n_d} w_{d;j} Z_{i,j}(\varkappa_i, \varkappa_j) \big\}$ and $R_w(\mathbf{r}) = \sum_{i=1}^{n_r} w_{r;i} R_i(r_i)$

Assigning weighting factors is useful only if the relevant contributions have been previously normalized so that they are all roughly the same magnitude. This serves as the baseline for which all weights are equal. The question immediately arises as to what criteria to use to weight the distance contributions.

We shall determine scaling factors, $Q_d > 0$ and $Q_r > 0$, so that the average distances per dimension of the discrete, range, and continuous contributions are equal, where by average we mean a utility-weighted average over the entire space of possible trades. This weighting places

[4] Later we shall generalize this distance to include weighting of dimensions.

[5] $\mathbf{M} = [m_{i,j}] = \mathrm{diag}(\mathbf{v})$ is the diagonal matrix formed by setting $m_{i,i} = v_i$ and $m_{i,j} = 0$ for $j \neq i$.

more emphasis on the better trades

If d_c, d_d, and d_r are the continuous, discrete, and range contributions to the total distance, then after multiplication by the scaling factors $d = d_c + Q_d d_d + Q_r d_r$. The scaling factors are determined through the utility weighted average distances defined by

$$\langle d_r \rangle \equiv \frac{\sum_{\boldsymbol{\varkappa}} \sum_r \int_V d\mathbf{u}\, Q_r d_r \exp[-Q_r d_r - Q_d d_d - d_c]}{\sum_{\boldsymbol{\varkappa}} \sum_r \int_V d\mathbf{u}\, \exp[-Q_r d_r - Q_d d_d - d_c]} = Q_r \frac{\sum_{\boldsymbol{\varkappa}} \exp[-Q_d d_d] \sum_r d_r \exp[-Q_r d_r] \int_V d\mathbf{u}\, \exp[-d_c]}{\sum_{\boldsymbol{\varkappa}} \exp[-Q_d d_d] \sum_r \exp[-Q_r d_r] \int_V d\mathbf{u}\, \exp[-d_c]}$$

$$\langle d_d \rangle \equiv \frac{\sum_{\boldsymbol{\varkappa}} \sum_r \int_V d\mathbf{u}\, Q_d d_d \exp[-Q_r d_r - Q_d d_d - d_c]}{\sum_{\boldsymbol{\varkappa}} \sum_r \int_V d\mathbf{u}\, \exp[-Q_r d_r - Q_d d_d - d_c]} = Q_d \frac{\sum_{\boldsymbol{\varkappa}} d_d \exp[-Q_d d_d] \sum_r \exp[-Q_r d_r] \int_V d\mathbf{u}\, \exp[-d_c]}{\sum_{\boldsymbol{\varkappa}} \exp[-Q_d d_d] \sum_r \exp[-Q_r d_r] \int_V d\mathbf{u}\, \exp[-d_c]}$$

$$\langle d_c \rangle \equiv \frac{\sum_{\boldsymbol{\varkappa}} \sum_r \int_V d\mathbf{u}\, d_c \exp[-Q_r d_r - Q_d d_d - d_c]}{\sum_{\mathbf{v}} \sum_r \int_V d\mathbf{u}\, \exp[-Q_r d_r - Q_d d_d - d_c]} = \frac{\sum_{\boldsymbol{\varkappa}} \exp[-Q_d d_d] \sum_r \exp[-Q_r d_r] \int_V d\mathbf{u}\, d_c \exp[-d_c]}{\sum_{\boldsymbol{\varkappa}} \exp[-Q_d d_d] \sum_r \exp[-Q_r d_r] \int_V d\mathbf{u}\, \exp[-d_c]}$$

A few comments on the above equations are in order. First, $\sum_{\boldsymbol{\varkappa}}$ indicates the repeated sum $\sum_{\varkappa_1} \cdots \sum_{\varkappa_{n_d}}$ over all possible discrete trades. $\sum_r$ indicates a sum over all the range variables and the integral over volume V indicates integration over the continuous trading volume of interest. Finally, we have not included a scaling factor Q_c on the continuous distance, since this can be made equal to 1 if we reinterpret Q_r as Q_r/Q_c and Q_d as Q_d/Q_c.[6] Each of the averages is an explicit function Q_d and Q_r.

The requirement on equal average contributions determines the two unknowns Q_r and Q_d through the equations: $\langle d_r \rangle / n_r = \langle d_c \rangle / n_c$ and $\langle d_d \rangle / n_d = \langle d_c \rangle / n_c$. These two nonlinear equations are coupled in terms of Q_r and Q_d and must be solved simultaneously for Q_r and Q_d. Further details are found in Appendix C.

6.1.3 Constraint Specification

Buyers and sellers may express constraints over both continuous and discrete variables.

Continuous Constraints

For simplicity (and because additional expressiveness is rarely required) we assume that the buyer's constraints over the continuous variables are linear.[7] This allows a buyer to express a constraint, e.g., the time of delivery must be within 10 days or I will not trade, i.e., $t \leq 10$. We allow for both inequality and equality constraints which can be expressed as $\mathbf{G}_1^{(b)}\mathbf{x} = \mathbf{g}_1^{(b)}$ and $\mathbf{G}_2^{(b)}\mathbf{x} \leq \mathbf{g}_2^{(b)}$. If there are $c_1^{(b)}$ equality constraints then $\mathbf{G}_1^{(b)}$ has $c_1^{(b)}$ rows. Similarly, $\mathbf{G}_2^{(b)}$ has $c_2^{(b)}$ rows if there are $c_2^{(b)}$ inequality constraints. We allow the constraints to depend on the setting of the discrete variables, and to be explicit we often write $\mathbf{G}_1^{(b)}(\boldsymbol{\varkappa})$, $\mathbf{g}_1^{(b)}(\boldsymbol{\varkappa})$, $\mathbf{G}_2^{(b)}(\boldsymbol{\varkappa})$, and $\mathbf{g}_2^{(b)}(\boldsymbol{\varkappa})$.

Discrete Constraints

We use a standard methodology to represent and process constraints over discrete variables [5]. Abstractly, a constraint over a (perhaps proper)[8] subset of the discrete variables is represented as a list of all the allowed combinations the variables may assume. An example representation of a pair of discrete constraints is given in Table 2. There are two solutions to this set of constraints: $\varkappa_1 = 1$, $\varkappa_2 = 2$, $\varkappa_3 = 3$ and $\varkappa_1 = 3$, $\varkappa_2 = 2$, $\varkappa_3 = 1$. We indicate these solutions as $\{\langle \varkappa_1, 1\rangle, \langle \varkappa_2, 2\rangle\}, \langle \varkappa_3, 3\rangle\}\}$ and $\{\langle \varkappa_1, 3\rangle, \langle \varkappa_2, 2\rangle\}, \langle \varkappa_3, 1\rangle\}\}$ respectively. Each solution where

[6] We singled the continuous variables out, since there will always be continuous variables in any trading scenario.

[7] With little additional complexity we can also handle quadratic constraints, see [4].

[8] A proper subset of a set is the set itself.

$\varkappa_1$	$\varkappa_2$	$\varkappa_3$
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

(a)

$\varkappa_2$
2

(b)

Table 2: An example set of constraints involving 3 variables where the domains of all variables are $\mathcal{D}_1 = \mathcal{D}_2 = \mathcal{D}_3 = \{1, 2, 3\}$. Constraint (a) requires that the values assumed by $\varkappa_1$, $\varkappa_2$, and $\varkappa_3$ are all different from each other, and constraint (b) requires that the value assumed by $\varkappa_2$ is even. See text for the solution to both constraints.

all the variables have been identified with specific values from their domains is called a labelling.

Computationally efficient representations are used to ensure that only feasible combinations of variables are ever processed. Numerous third-party libraries offer constraint programming functionality.[9]

6.1.4 Utility and Total Cost of Ownership

The buyer's utility function and associated constraints may be difficult for many users to define. In this section we show how models of the buyer's business can be used to define utility in a natural manner.

We imagine a function which provides an estimate of the total cost of ownership for any given purchase. Cost contributions to this function might include piece part costs, freight costs, setup costs, quality assurance costs, repair costs, etc. It is important to include all quantifiable costs associated with the lifetime of use of the purchased product because it is this function we will be minimizing. Significant savings may be obtained by taking a longer-term view of the purchase. Revenues (negative costs) generated from the purchase are also included in the function so that the function represents some measure of profitability associated with the purchase. We write the total cost of ownership function as $C_o(\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r}; \boldsymbol{\beta})$. We explicitly indicate the dependence on the negotiated trade parameters $\mathbf{x}$, $\boldsymbol{\varkappa}$, and $\mathbf{r}$, as well as other factors $\boldsymbol{\beta}$. The other factors might include forecasted demand, current inventory levels, etc. These factors will vary over time, and they can be extracted from the buyer's ERP and supply chain management systems (SCM) in real-time just before the purchase to ensure continuous real-time optimization. See section 6.2.1 for further details.

Minimization of $C_o(\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r}; \boldsymbol{\beta})$ defines an ideal trade dependent on current conditions: $\mathbf{x}_{\text{opt}}(\boldsymbol{\beta})$, $\boldsymbol{\varkappa}_{\text{opt}}(\boldsymbol{\beta})$, $\mathbf{r}_{\text{opt}}(\boldsymbol{\beta})$. If desired, these can be used to define $\boldsymbol{\mu} = \mathbf{x}_{\text{opt}}(\boldsymbol{\beta})$, $r = \mathbf{r}_{\text{opt}}(\boldsymbol{\beta})$ and the desired ideal discrete configuration $\boldsymbol{\varkappa}_{\text{opt}}(\boldsymbol{\beta})$ (having distance contribution $Z = 0$). Moreover, the tradeoffs between continuous dimensions around this minimum can be obtained

[9] See for example `www.mozart-oz.org` or `www.ilog.com`.

through calculation of the Hessian matrix $\mathbf{H} = [h_{i,j}]$ where the i, j matrix element is given by

$$h_{i,j} = \left. \frac{\partial^2 C_o(\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r}; \boldsymbol{\beta})}{\partial x_i \partial x_j} \right|_{\mathbf{x}=\mathbf{x}_{\text{opt}}(\boldsymbol{\beta}), \boldsymbol{\varkappa}=\boldsymbol{\varkappa}_{\text{opt}}(\boldsymbol{\beta}), \mathbf{r}=\mathbf{r}_{\text{opt}}(\boldsymbol{\beta})}$$

We then identify $\mathbf{C}^{-1}$ with $\mathbf{H}$. In this way, little trading flexibility is obtained in directions where total cost of ownership rises rapidly, while significant flexibility is obtained in directions where total cost of ownership increases slowly.

In summary, a total cost of ownership model defines both the most preferred trade parameters and the flexibility possessed around the preferred trade. The model pulls dynamically from real-time data sources to provide the most up-to-date optimization based on total costs of ownership and other important qualitative factors the buyer may wish to describe in the utility function. The same function and its constituent costs may also be used to help analyze proposed trades from suppliers.

6.1.5 Supplier Capabilities

As discussed in the introduction, suppliers represent their capabilities through a specification of the subspace of X in which they will trade. A supplier's capabilities must specify the allowed continuous, discrete, and range variables. The allowed range variables are expressed as the pairs $(\underline{r}_j, \overline{r}_j)$, one for each range variable. For example, if a supplier produces 25mm inner diameter screws to within a tolerance of 0.5 mm, then the range variable is simply $(24.5, 25.5)$. These are compared with the buyer's ideal range and contribute to the distance function through the $\mathbf{R}_w(\boldsymbol{\varkappa})$ function.

Capabilities over continuous and discrete variables are more complex.

Continuous Capabilities

Continuous capabilities are viewed naturally as responses to a buyer's request. Thus we distinguish between a buyer's requested continuous vector $\mathbf{x}^{(b)}$ and a seller's response $\mathbf{x}^{(s)}$. A vector-valued function, $\mathbf{f}(\mathbf{x}^{(b)}, \mathbf{x}^{(s)}, \boldsymbol{\varkappa})$ returns the response based on the buyer's request and also, perhaps, other previously defined supplier responses. Component f_i of $\mathbf{f}$ defines the ith continuous variable, *i.e.* $x_i^{(s)} = f_i(\mathbf{x}^{(b)}, \mathbf{x}^{(s)})$.

Currently, suppliers are used to quoting price discounts for large volume orders and these price discounts are expressed as piecewise linear functions. Consequently, we restrict f_i to have the following form (where we distinguish between the functions depending on the buyer and seller variables):

$$x_i^{(s)} = \sum_k f_{i,k}^{(s)}(x_k^{(s)}, \boldsymbol{\varkappa}^{(s)}) + f_{i,k}^{(b)}(x_k^{(b)}, \boldsymbol{\varkappa}^{(s)}). \tag{2}$$

An example of how this may be used to define a supplier response is the following: We assume three continuous dimensions – price, volume, and time of delivery and indicate these

as $[x_1, x_2, x_3] = [p, v, t]$. Then a response may be formed as

$$
\begin{aligned}
v^{(s)} &= f_{v,v}(v^{(b)}) \\
t^{(s)} &= f_t(v^{(s)}, t^{(b)}) = f_{t,v}(v^{(s)}) \\
p^{(s)} &= f_p(v^{(s)}, t^{(s)}) = f_{p,v}(v^{(s)}) + f_{p,t}(t^{(s)}).
\end{aligned}
$$

The $f_{v,v}$ function returns the volume a supplier will fulfill as a function of what the buyer asked for. If the supplier can deliver any volume, this will be the identity function. If the supplier delivers only in certain lot sizes, this function may have a staircase shape, etc. The $f_{t,v}$ function indicates the time it will take a supplier to deliver a certain volume. So, for example, if larger shipments require longer transportation, then this dependence is given by this function. Finally, we turn to the price determination. In this example the price depends on the quantity $v^{(s)}$ being shipped and the $f_{p,v}$ might represent price discounts for large volume orders. There is also an incremental price contribution based on the time of delivery. If faster delivery is more expensive, this is reflected in $f_{p,t}$.

For a given setting of the discrete variables, each $f_{i,k}^{(s)}(x_k^{(s)}, \boldsymbol{\varkappa}^{(s)})$ and $f_{i,k}^{(b)}(x_k^{(b)}, \boldsymbol{\varkappa}^{(s)})$is a one-dimensional piecewise linear function. Consequently, the functions can be specified by listing the breakpoints. If $f_{i,k}^{(s)}(x_k^{(s)}, \boldsymbol{\varkappa}^{(s)})$ has $\kappa_{i,k}^{(s)}$ breakpoints, then we list these as $\{x_k^{(s)}(b_{i,k}^{(s)} = 1), x_k^{(s)}(b_{i,k}^{(s)} = 2), \cdots, x_k^{(s)}(b_{i,k}^{(s)} = \kappa_{i,k}^{(s)})\}$[10] and function values at these breakpoints are $\{f_{i,k}^{(s)}(x_k^{(s)}(b_{i,k}^{(s)} = 1)), f_{i,k}^{(s)}(x_k^{(s)}(b_{i,k}^{(s)} = 2)), \cdots, f_{i,k}^{(s)}(x_k^{(s)}(b_{i,k}^{(s)} = \kappa_{i,k}^{(s)}))\}$. Similarly, $f_{i,k}^{(b)}$, is a piecewise linear function defined by the $\kappa_{i,k}^{(b)}$ breakpoints $\{x_k^{(b)}(b_{i,k}^{(b)} = 1), x_k^{(b)}(b_{i,k}^{(b)} = 2), \cdots, x_k^{(b)}(b_{i,k}^{(b)} = \kappa_{i,k}^{(b)})\}$ and function values at these breakpoints $\{f_{i,k}^{(b)}(x_k^{(b)}(b_{i,k}^{(b)} = 1)), f_{i,k}^{(b)}(x_k^{(b)}(b_{i,k}^{(b)} = 2)), \cdots, f_{i,k}^{(b)}(x_k^{(b)}(b_{i,k}^{(b)} = \kappa_{i,k}^{(b)}))\}$. The breakpoints are indexed by the integers $b_{i,k}^{(s)}$ and $b_{i,k}^{(b)}$.

An interval is specified by assigning a value $b_{i,k}^{(s)} \in [1, \kappa_{i,k}^{(s)} - 1]$ and $b_{i,k}^{(b)} \in [1, \kappa_{i,k}^{(b)} - 1]$ so that[11]

$$
x_k^{(s)}(b_{i,k}^{(s)}) \le x_k^{(s)} \le x_k^{(s)}(b_{i,k}^{(s)} + 1) \ \forall\, i \quad \text{and} \quad x_k^{(b)}(b_{i,k}^{(b)}) \le x_k^{(b)} \le x_k^{(b)}(b_{i,k}^{(b)} + 1) \ \forall\, i. \tag{3}
$$

Within each interval the functions are linear, so we have

$$
x_i^{(s)} = c_i(\boldsymbol{\varkappa}^{(s)}, \{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\}) + \sum_k m_{i,k}^{(s)}(\boldsymbol{\varkappa}^{(s)}, b_{i,k}^{(s)}) x_k^{(s)} + m_{i,k}^{(b)}(b_{i,k}^{(b)}) x_k^{(b)}
$$

where $c_i(\mathbf{v}^{(s)}, \{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\}) = \sum_k c_{i,k}^{(s)}(\boldsymbol{\varkappa}^{(s)}, b_{i,k}^{(s)}) + c_{i,k}^{(b)}(b_{i,k}^{(b)})$. In the above equations, the intercepts and slopes are given explicitly by

$$
c_{i,k}^{(s)}(b_{i,k}^{(s)}) = \frac{x_k^{(s)}(b_{i,k}^{(s)} + 1) f_{i,k}^{(s)}(x_k^{(s)}(b_{i,k})) - x_k^{(s)}(b_{i,k}) f_{i,k}^{(s)}(x_k^{(s)}(b_{i,k} + 1))}{x_k^{(s)}(b_{i,k}^{(s)} + 1) - x_k^{(s)}(b_{i,k})}
$$

[10] The breakpoints are assumed to be in increasing order.

[11] Note that many choices for $\{b_{i,k}^{(s)}\}$ or $\{b_{i,k}^{(b)}\}$ would be inconsistent with these constraints. A simple way to fix this is to define a union of all breakpoints (s) and (b), and have all $f_{i,k}$ have the same breakpoints.

and

$$m_{i,k}^{(s)}(b_{i,k}^{(s)}) = \frac{f_{i,k}^{(s)}\left(x_k^{(s)}(b_{i,k}^{(s)}+1)\right) - f_{i,k}^{(s)}\left(x_k^{(s)}(b_{i,k})\right)}{x_k^{(s)}(b_{i,k}^{(s)}+1) - x_k^{(s)}(b_{i,k}^{(s)})}$$

respectively. An analogous result holds for the $c_{i,k}^{(b)}(b_{i,k}^{(b)})$ and $m_{i,k}^{(b)}(b_{i,k}^{(b)})$.

To eliminate any cyclic dependence on $x_i^{(s)}$ we must impose an ordering on $x_i^{(s)}$ so that $x_i^{(s)}$ can only depend on $x_j^{(s)}$ where $j < i$. Consequently, we can write

$$x_i^{(s)} = c_i(\boldsymbol{\pi}^{(s)}, \{b_{i,1}^{(s)}, \cdots, b_{i,i-1}^{(s)}\}, \{b_{i,k}^{(b)}\}) + \sum_{k<i} m_{i,k}^{(s)}(\boldsymbol{\pi}^{(s)}, b_{i,k}^{(s)}) x_k^{(s)} + \sum_k m_{i,k}^{(b)}(b_{i,k}^{(b)}) x_k^{(b)}.$$

Written as a matrix equation, the above becomes

$$\left(\mathbf{I} - \mathbf{M}^{(s)}(\boldsymbol{\pi}^{(s)}, \{b_{i,k}^{(s)}\})\right)\mathbf{x}^{(s)} = \mathbf{c}(\boldsymbol{\pi}^{(s)}, \{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\}) + \mathbf{M}^{(b)}(\{b_{i,k}^{(b)}\})\mathbf{x}^{(b)}$$

where $\mathbf{c}(\boldsymbol{\pi}^{(s)}, \{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\}) = \left[c_1(\boldsymbol{\pi}^{(s)}, \{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\}) \quad \cdots \quad c_n(\boldsymbol{\pi}^{(s)}, \{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\})\right]^t$, $\mathbf{x}^{(s)} = \left[x_1^{(s)} \quad \cdots \quad x_{n_c}^{(s)}\right]^t$, $\mathbf{x}^{(b)} = \left[x_1^{(b)} \quad \cdots \quad x_{n_c}^{(b)}\right]^t$, and

$$\mathbf{M}^{(s)}(\boldsymbol{\pi}^{(s)}, \{b_{i,k}^{(s)}\}) = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ m_{2,1}^{(s)}(\boldsymbol{\pi}^{(s)}, b_{2,1}^{(s)}) & 0 & \cdots & 0 & 0 \\ m_{3,1}^{(s)}(\boldsymbol{\pi}^{(s)}, b_{3,1}^{(s)}) & m_{3,2}^{(s)}(\boldsymbol{\pi}^{(s)}, b_{3,2}^{(s)}) & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ m_{n,1}^{(s)}(\boldsymbol{\pi}^{(s)}, b_{n,1}^{(s)}) & m_{n,2}^{(s)}(\boldsymbol{\pi}^{(s)}, b_{n,2}^{(s)}) & \cdots & m_{n,n-1}^{(s)}(\boldsymbol{\pi}^{(s)}, b_{n,n-1}^{(s)}) & 0 \end{bmatrix}$$

and

$$\mathbf{M}^{(b)}(\{b_{i,k}^{(b)}\}) = \begin{bmatrix} m_{1,1}^{(b)}(b_{1,1}^{(b)}) & m_{1,2}^{(b)}(b_{1,2}^{(b)}) & \cdots & m_{1,n}^{(b)}(b_{1,n}^{(b)}) \\ m_{2,1}^{(b)}(b_{2,1}^{(b)}) & m_{2,2}^{(b)}(b_{2,2}^{(b)}) & \cdots & m_{2,n}^{(b)}(b_{2,n}^{(b)}) \\ \vdots & \vdots & \ddots & \vdots \\ m_{n,1}^{(b)}(b_{n,1}^{(b)}) & m_{n,2}^{(b)}(b_{n,2}^{(b)}) & \cdots & m_{n,n}^{(b)}(b_{n,n}^{(b)}) \end{bmatrix}.$$

In most cases $\mathbf{x}^{(s)}$ will depend only on a subset of the variables in $\mathbf{x}^{(b)}$. If $\mathbf{x}^{(s)}$ depends on $n' \leq n$ of the $\mathbf{x}^{(b)}$ variables, then $\mathbf{M}^{(b)}$ is an $n \times n'$ matrix. In the example given everything depending only upon the volume the buyer requested.

Since $\mathbf{M}^{(s)}(\boldsymbol{\pi}^{(s)})$ is lower triangular and can be inverted in time $\mathcal{O}(n)$, we can rapidly express $\mathbf{x}^{(s)}$ as

$$\mathbf{x}^{(s)} = \left(\mathbf{I} - \mathbf{M}^{(s)}(\boldsymbol{\pi}^{(s)}, \{b_{i,k}^{(s)}\})\right)^{-1}\mathbf{c}(\boldsymbol{\pi}^{(s)}, \{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\}) + \left(\mathbf{I} - \mathbf{M}^{(s)}(\boldsymbol{\pi}^{(s)}, \{b_{i,k}^{(s)}\})\right)^{-1}\mathbf{M}^{(b)}(\{b_{i,k}^{(b)}\})\mathbf{x}^{(b)} \tag{4}$$

as long as the $b_{i,k}^{(s)}$ are chosen to also satisfy $x_k^{(s)}(b_{i,k}^{(s)}) \leq x_k^{(s)} \leq x_k^{(s)}(b_{i,k}^{(s)}+1)$. These constraints will be used in section 6.1.6 which formulates the optimization problem.

We also allow a supplier to express additional linear constraints so that, for example, he may represent that he does not deliver on Sunday. Thus the supplier may define the

matrices $\mathbf{G}_1^{(s)}(\boldsymbol{\varkappa})$, $\mathbf{G}_2^{(s)}(\boldsymbol{\varkappa})$, and the vectors $\mathbf{g}_1^{(s)}(\boldsymbol{\varkappa})$, $\mathbf{g}_2^{(s)}(\boldsymbol{\varkappa})$ such that $\mathbf{G}_1^{(s)}\mathbf{x}^{(s)} = \mathbf{g}_1^{(s)}$ and $\mathbf{G}_2^{(s)}\mathbf{x}^{(s)}\mathbf{x} \leq \mathbf{g}_2^{(s)}$. $\mathbf{G}_1^{(s)}(\boldsymbol{\varkappa})$ and $\mathbf{G}_2^{(s)}(\boldsymbol{\varkappa})$ have $c_1^{(s)}$ and $c_2^{(s)}$ rows respectively.

Discrete Capabilities

It is easy to imagine that a supplier's response on a discrete dimension is highly constrained by the values of the response on other dimensions, e.g., certain product characteristics come only in certain colors and package sizes. Consequently, it is not suitable to explicitly define a response but only to make available a supplier's constraints amongst the discrete variables. Consider 3 discrete dimensions where $\varkappa_1 \in \mathcal{D}_1 = [a, b, c]$, $\varkappa_2 \in \mathcal{D}_2 = [A, B, C, D]$, and $\varkappa_3 \in \mathcal{D}_3 = [\alpha, \beta, \gamma, \delta]$, and assume the supplier has the following 3 constraints

$$\begin{aligned} \mathcal{C}_1(\varkappa_1, \varkappa_3) &= \{(a, \alpha), (a, \beta), (b, \beta), (c, \beta)\}, \\ \mathcal{C}_2(\varkappa_2, \varkappa_3) &= \{(A, \beta), (B, \gamma), (D, \beta)\}, \\ \mathcal{C}_3(\varkappa_1) &= \{b, c\}. \end{aligned}$$

We first note that there are 4 feasible solutions (or product configurations the supplier can meet): $[\varkappa_1, \varkappa_2, \varkappa_3] = [b, A, \beta]$, $[b, D, \beta]$, $[c, A, \beta]$, or $[c, D, \beta]$. Feasible solutions to the constraints define the response $\boldsymbol{\varkappa}^{(s)}$ for the discrete variables.

We indicate a supplier's or buyer's collective set of discrete constraints by $\mathcal{C}^{(s)}(\boldsymbol{\varkappa})$ and $\mathcal{C}^{(b)}(\boldsymbol{\varkappa})$ respectively.

6.1.6 The Optimization Problem

Having defined the necessary components, we now define the optimization task which determines the continuous $\mathbf{x}^*$ and discrete $\boldsymbol{\varkappa}^*$ parameters of the trade.

Since the trade must be acceptable to the supplier, we maximize the buyer's utility over a supplier's capabilities. Equivalently, we minimize the distance from the buyer's ideal values as

$$[\mathbf{x}^*, \boldsymbol{\varkappa}^*] = \arg\min_{\mathbf{x}^{(s)}, \boldsymbol{\varkappa}^{(s)}} \left\{ \left(\mathbf{x}^{(s)} - \boldsymbol{\mu}(\boldsymbol{\varkappa}^{(s)})\right)^t \mathbf{C}_w^{-1}(\boldsymbol{\varkappa}^{(s)}) \left(\mathbf{x}^{(s)} - \boldsymbol{\mu}(\boldsymbol{\varkappa}^{(s)})\right) + Q_d Z_w(\boldsymbol{\varkappa}^{(s)}) + Q_r R_w(\mathbf{w}) \right\}$$

where

$$\mathbf{x}^{(s)} = \left(\mathbf{I} - \mathbf{M}^{(s)}(\boldsymbol{\varkappa}^{(s)})\right)^{-1} \mathbf{c}(\boldsymbol{\varkappa}^{(s)}) + \left(\mathbf{I} - \mathbf{M}^{(s)}(\boldsymbol{\varkappa}^{(s)})\right)^{-1} \mathbf{M}^{(b)} \mathbf{x}^{(b)}$$

subject to the constraints over continuous variables

$$\mathbf{G}_1(\boldsymbol{\varkappa}^{(s)})\mathbf{x}^{(s)} = \mathbf{g}_1(\boldsymbol{\varkappa}^{(s)}), \qquad \mathbf{G}_2(\boldsymbol{\varkappa}^{(s)})\mathbf{x}^{(s)} \leq \mathbf{g}_2(\boldsymbol{\varkappa}^{(s)})$$

and the constraints over the discrete variables $\mathcal{C}^{(b)}(\mathbf{v}^{(s)})$, $\mathcal{C}^{(s)}(\mathbf{v}^{(s)})$. In the above, we have defined the $(c_1^{(s)} + c_1^{(b)}) \times n_c$ and $(c_2^{(s)} + c_2^{(b)}) \times n_c$ matrices $\mathbf{G}_1(\boldsymbol{\varkappa}^{(s)})$ and $\mathbf{G}_2(\boldsymbol{\varkappa}^{(s)})$ by

$$\mathbf{G}_1(\boldsymbol{\varkappa}^{(s)}) = \begin{bmatrix} \mathbf{G}_1^{(s)}(\boldsymbol{\varkappa}^{(s)}) \\ \mathbf{G}_1^{(b)}(\boldsymbol{\varkappa}^{(s)}) \end{bmatrix}, \quad \text{and} \quad \mathbf{G}_2(\boldsymbol{\varkappa}^{(s)}) = \begin{bmatrix} \mathbf{G}_2^{(s)}(\boldsymbol{\varkappa}^{(s)}) \\ \mathbf{G}_2^{(b)}(\boldsymbol{\varkappa}^{(s)}) \end{bmatrix}.$$

The $(c_1^{(s)} + c_1^{(b)})$- and $(c_1^{(s)} + c_1^{(b)})$-vectors $\mathbf{g}_1(\boldsymbol{\varkappa}^{(s)})$ and $\mathbf{g}_2(\boldsymbol{\varkappa}^{(s)})$ are defined by

$$\mathbf{g}_1(\boldsymbol{\varkappa}^{(s)}) = \begin{bmatrix} \mathbf{g}_1^{(s)}(\boldsymbol{\varkappa}^{(s)}) \\ \mathbf{g}_1^{(b)}(\boldsymbol{\varkappa}^{(s)}) \end{bmatrix}, \quad \text{and} \quad \mathbf{g}_2(\boldsymbol{\varkappa}^{(s)}) = \begin{bmatrix} \mathbf{g}_2^{(s)}(\boldsymbol{\varkappa}^{(s)}) \\ \mathbf{g}_2^{(b)}(\boldsymbol{\varkappa}^{(s)}) \end{bmatrix}.$$

The optimization is accomplished by iterating two distinct phases. Phase one sets the continuous parameters optimally for a given setting of the discrete variables. We define the functions

$$d_1(\mathbf{x}, \boldsymbol{\varkappa}) = \big(\mathbf{x} - \boldsymbol{\mu}(\boldsymbol{\varkappa})\big)^t \mathbf{C}_w^{-1}(\boldsymbol{\varkappa})\big(\mathbf{x} - \boldsymbol{\mu}(\boldsymbol{\varkappa})\big) + R(\mathbf{r}; \boldsymbol{\varkappa}) \quad \text{and} \quad d_2(\boldsymbol{\varkappa}) = Q_d Z_w(\boldsymbol{\varkappa}^{(s)}),$$

The first phase of the optimization is the continuous problem:[12]

$$\mathbf{x}^*(\boldsymbol{\varkappa}) = \arg\min_{\mathbf{x}^{(s)}} d_1(\mathbf{x}^{(s)}, \boldsymbol{\varkappa}) \quad \text{subject to} \quad \mathbf{G}_1(\boldsymbol{\varkappa})\mathbf{x} = \mathbf{g}_1(\boldsymbol{\varkappa}) \text{ and } \mathbf{G}_2(\boldsymbol{\varkappa})\mathbf{x} \leq \mathbf{g}_2(\boldsymbol{\varkappa}). \tag{5}$$

A detailed discussion on the solution of the phase 1 optimization problem is found in appendix D. The second phase determines the best value of the discrete variables by minimizing over a function of $\boldsymbol{\varkappa}$ alone

$$\boldsymbol{\varkappa}^* = \arg\min_{\boldsymbol{\varkappa}} d_1(\mathbf{x}(\boldsymbol{\varkappa}), \boldsymbol{\varkappa}) + d_2(\boldsymbol{\varkappa}) \quad \text{subject to} \quad \mathcal{C}^{(b)}(\boldsymbol{\varkappa}) \wedge \mathcal{C}^{(s)}(\boldsymbol{\varkappa}). \tag{6}$$

Further details on the phase 2 optimization are given in Appendix E. Once $\boldsymbol{\varkappa}^*$ has been determined, we find $\mathbf{x}^*$ as $\mathbf{x}^* = \mathbf{x}(\boldsymbol{\varkappa}^*)$.

6.1.7 Aggregation

Often a buyer may be willing to divide an order between multiple suppliers in order to aggregate the required demand or to obtain better deals. In this section, we detail how the present invention supports this aggregate optimization.

Aggregation can only occur over the continuous variables where values may be subdivided. Each continuous variable x_i must be parcelled out amongst a set of suppliers. Consequently, we extend our notation to $x_i \to \tilde{x}_{i,k}$ giving the contribution of the kth supplier to continuous dimension i. The kth supplier may come from a (perhaps proper) subset of all suppliers. We indicate the set of potentially contributing suppliers as $\mathcal{K}$ and the number of potentially contributing suppliers as $|\mathcal{K}|$.[13]

We restrict the discrete variables to be the same across all potentially aggregated suppliers, i.e., we do *not* generalize $\varkappa_i \to \varkappa_{i,k}$. This simplifying assumption is made for two reasons. First, the size of the discrete optimization problem is smaller and so optimization be performed faster. Second, it may be difficult to elicit from the buyer the allowed discrete alternatives for each supplier. Nevertheless, this generalization is straightforward should the need arise.

This simplifying assumption requires that the union of discrete supplier constraints $\mathcal{C}_{\mathcal{K}}(\boldsymbol{\varkappa}) \equiv$

[12] No optimization is required over range variables, since these are specified up front by both buyer and seller and merely add to the total distance.

[13] All work to this point is thus seen as the special case $|\mathcal{K}| = 1$.

$\bigwedge_{k\in\mathcal{K}} \mathcal{C}_k^{(s)}(\boldsymbol{\varkappa})$ yields a feasible solution when combined with the buyer's discrete constraints $\mathcal{C}^{(b)}(\boldsymbol{\varkappa})$. A necessary (but not sufficient) condition for satisfaction is then that each constraint satisfaction problem k having constraints $\mathcal{C}^{(b)}(\boldsymbol{\varkappa}) \wedge \mathcal{C}_k^{(s)}(\boldsymbol{\varkappa})$ has a feasible solution.[14] Henceforth, we will assume that the set of suppliers, $\mathcal{K}$, satisfies this condition. If not, those suppliers violating the constraints $\mathcal{C}^{(b)}(\boldsymbol{\varkappa}) \wedge \mathcal{C}_k^{(s)}(\boldsymbol{\varkappa})$ are eliminated from consideration in $\mathcal{K}$.

Discrete Search

We must search over the subsets of $\mathcal{K}$ for feasible solutions, which is a combinatorial problem. Fortunately, given a complete labelling of variables, determining the largest subset is easy. For any given labelling of all discrete variables, if each $\mathcal{C}^{(b)} \wedge \mathcal{C}_k^{(s)}\ \forall\ k \in \kappa \subset \mathcal{K}$ is satisfiable, then the union $\mathcal{C}^{(b)} \wedge \mathcal{C}_\kappa^{(s)}$ where $\mathcal{C}_\kappa^{(s)} = \bigwedge_{k\in\kappa} \mathcal{C}_k^{(s)}$ is also satisfiable under the same labelling. The largest subset of variables is found by adding all k which have feasible solutions with the buyer. We needn't worry about smaller subsets because the continuous optimization will assign zero values to those if appropriate. Consequently, for any given labelling $\boldsymbol{\varkappa}$ we let $\mathcal{K}(\boldsymbol{\varkappa})$ represent the maximal subset of suppliers for which $\mathcal{C}^{(b)}(\boldsymbol{\varkappa}) \wedge \mathcal{C}_\mathcal{K}(\boldsymbol{\varkappa})$ is satisfiable. It is this set of suppliers which enter into the continuous optimization. The number of participating suppliers is denoted by $|\mathcal{K}(\boldsymbol{\varkappa})|$.

Continuous Optimization

Optimization over the continuous variables is carried for each labelling $\boldsymbol{\varkappa}$. Generally speaking, the buyer's utility will not be an explicit function of $x_{i;k}$ but only of x_i. We assume a linear relationship between these two quantities so that[15]

$$\mathbf{x} = \boldsymbol{\Xi}\tilde{\mathbf{x}}.$$

The $n_c|\mathcal{K}(\boldsymbol{\varkappa})|$ vector $\tilde{x}$ is defined as $\tilde{\mathbf{x}}^t = [\tilde{\mathbf{x}}_1, \cdots, \tilde{\mathbf{x}}_{n_c}]$ where $\tilde{\mathbf{x}}_i^t = [\tilde{x}_{i;1}, \cdots, \tilde{x}_{i;|\mathcal{K}(\boldsymbol{\varkappa})|}]$. The $n_c \times n_c|\mathcal{K}(\boldsymbol{\varkappa})|$ matrix $\boldsymbol{\Xi}$ $\xi_{i,k}$ is assumed known and typically has the form[16]

$$\boldsymbol{\Xi} = \begin{bmatrix} \boldsymbol{\xi}_1^t & \mathbf{0}^t & \cdots & \mathbf{0}^t \\ \mathbf{0}^t & \boldsymbol{\xi}_2^t & \cdots & \mathbf{0}^t \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}^t & \mathbf{0}^t & \cdots & \boldsymbol{\xi}_{n_c}^t \end{bmatrix}$$

where $\mathbf{0}$ is the K-vector of all zeros and $\boldsymbol{\xi}_i$ is the linear combination relating x_i to the $\tilde{x}_{i;k}$. Under our assumptions for $\boldsymbol{\Xi}$, $x_i = \boldsymbol{\xi}_i^t\tilde{\mathbf{x}}_i$. In cases where the buyer wants to accumulate the results from suppliers (e.g., aggregating quantities) $\boldsymbol{\xi} = \mathbf{1}$ is the $|\mathcal{K}(\boldsymbol{\varkappa})|$-vector of all 1s. In other cases the buyer may take $\boldsymbol{\xi} = \mathbf{1}/|\mathcal{K}(\boldsymbol{\varkappa})|$ so that the time of delivery becomes the average

[14] It is easily seen this requirement is not sufficient by having $\mathcal{C}^{(b)} = \{\langle\varkappa_1, 1\rangle, \langle\varkappa_1, 2\rangle\}$, $\mathcal{C}_1^{(s)} = \{\langle\varkappa_1, 1\rangle\}$, and $\mathcal{C}_2^{(s)} = \{\langle\varkappa_1, 2\rangle\}$.

[15] We can also relate $\mathbf{x}$ and $\tilde{\mathbf{x}}$ by $\mathbf{x} = \boldsymbol{\Xi}\tilde{\mathbf{x}}+\boldsymbol{\nu}$ for some constant vector $\boldsymbol{\nu}$, which will not cause any complications. However, there seems to be little practical reason to do so.

[16] With no additional computational complexity, we can allow $\boldsymbol{\Xi}$ to depend on $\boldsymbol{\varkappa}$.

time of delivery across the suppliers. Constraints on $\mathbf{x}$ become constraints on $\tilde{\mathbf{x}}$ via

$$\tilde{\mathbf{G}}_1(\varkappa)\tilde{\mathbf{x}} = \mathbf{g}_1(\varkappa) \quad \text{and} \quad \tilde{\mathbf{G}}_2(\varkappa)\tilde{\mathbf{x}} \leq \mathbf{g}_2(\varkappa) \tag{7}$$

where $\tilde{\mathbf{G}}_1(\varkappa) = \tilde{\mathbf{G}}_1(\varkappa)\Xi$ and $\tilde{\mathbf{G}}_1(\varkappa) = \tilde{\mathbf{G}}_1(\varkappa)\Xi$. We might also expect the buyer to add additional linear constraints, such as requiring the latest shipment from any supplier to arrive earlier than a certain date, or requiring all deliveries to arrive the same day. There can also be constraints specific to particular suppliers, e.g., the buyer doesn't want any more than 100 units from supplier 5. These can be handled simply as constraints on the individual $\tilde{x}_{i;k}$ and added as extra rows to $\tilde{\mathbf{G}}_1(\varkappa)$, $\tilde{\mathbf{G}}_2(\varkappa)$, $\tilde{\mathbf{g}}_1(\varkappa)$, and $\tilde{\mathbf{g}}_2(\varkappa)$.

With aggregation, the quadratic form to be minimized is $\left(\Xi\tilde{\mathbf{x}} - \boldsymbol{\mu}(\varkappa)\right)^t \mathbf{C}_w^{-1}(\varkappa)\left(\Xi\tilde{\mathbf{x}} - \boldsymbol{\mu}(\varkappa)\right)$ subject to the constraints given in Eq. (7). This minimization can be carried out through a straightforward generalization of the method given in Appendix D.

6.2 Implementation

In this section we outline an implementation of the entire e-procurement invention. We begin with a high-level description of the architecture, then fill in the details by describing a complete object model.

6.2.1 High-level Architecture of the Invention

There are at least two modes in which the invention may be used. First, the invention may reside at the site of large buyers, and suppliers who wish to sell to the buyer may be required to submit their capabilities via a web interface to the buyer. The invention may also be used within a marketplace hosted by a third party. Buyers/sellers log onto the market, submit their preference/capabilities, and act on the results. The architecture is modular enough to support both modes of operation.

In Figure 1 we present an architecture for the invention. We describe the architecture, starting from the optimization algorithm which finds matches between buyers and sellers and work our way outwards.

A controller surrounds the optimization engine, feeding it buyer preferences and seller capabilities. If multiple optimization processes are running (perhaps on different machines), the controller can also do load balancing, forwarding the request to the least busy process. The controller decomposes preferences and capabilities into their constituent buyer- and seller-specific versions (see below), selects the most specific matching preference/capability pairs, and sends them to the matching engine for optimization. The controller then collects responses from the matching engine and returns them to the buyer. Additionally, the controller logs all results into a database for recording purposes.

Another layer, called the Connector in Figure 1, separates the graphical user interface (GUI) through which users communicate with the tool from the controller. This layer serves a number of functions. The connector transforms the description of preferences and capabilities from the GUI into a form suitable for the implementation of the matching engine. Part of

this transformation involves validation of appropriate input from the GUI layer so that no malformed input is ever sent to the controller. The Connector layer can also pull data from ERP or SCM systems and automatically infer preferences (using the total cost of ownership function) for the buyer. The enterprise abstraction layer insulates the Connector from the precise details of the manner in which the ERP and SCM data needs to be gathered. Total cost of ownership is evaluated in the simulation modules, which may either be running locally at the client's site or running centrally at the main server. These simulation modules pull operational data (the vector β)from the enterprise abstraction layer. A preference optimization module (TCO) minimizes the total cost of ownership to determine the ideal trade and the flexibilities around the ideal trade.

At the outmost level, a layer provides integration with the GUI and/or host system. A number of administrative systems are expected at this layer. Market administration services allow easy definition of trading spaces, the dimensions of negotiation, limits on continuous variables, allowed settings of the discrete variables, etc. User administration services allow an administrator to define buyers, passwords, spending limits, etc. Supplier services accomplish similar tasks on the supply side. Managers for preferences, capabilities, and match results ensure that these objects are properly stored in a database. This layer layer also dynamically generates the html necessary for presentation of the data via a web interface to buyers and sellers.

For maximal portability, communications between the View and Connector are via XML documents. For maximal efficiency, communications between the Connector and matching controller are as serialized Java objects.

6.2.2 An Object Model for the Invention

The fundamental objects required for the invention are preferences from buyers, capabilities from sellers, and match results returned to all parties. The components of such objects have already been considered from a mathematical point of view, and we now describe one possible computer representation.

In this section we describe a complete grammar for the object model. The following syntactic conventions are used:

- ⟨nt⟩ denotes a non-terminal symbol nt
- [obj] denotes an optional grammar segment obj
- {obj} denotes 1, or many times the grammar segment obj
- $\rightarrow$ denotes a production rule for non-terminal symbol. If there are multiple rules, say ⟨a⟩, ⟨b⟩, and ⟨c⟩, then these are denoted as

$$\langle \text{nt} \rangle \rightarrow \langle a \rangle | \langle b \rangle | \langle c \rangle.$$

In contrast, a production rule of the form

$$\langle \text{nt} \rangle \rightarrow \langle a \rangle, \langle b \rangle, \langle c \rangle$$

indicates that the non-terminal ⟨nt⟩ is composed of three grammar segments, ⟨a⟩, ⟨b⟩, and ⟨c⟩

- terminal keywords are in serif font

Obvious non-terminal grammar elements like ⟨string⟩ and ⟨integer⟩ are not described.

Supply Side

To represent capabilities that apply to a specific buyer (perhaps for contractual reasons), we have defined a capability to be a list of ⟨buyerSpecificCapability⟩. With one exception, a buyer-specific capability applies only to one buyer – that buyer associated in the id field of the ⟨buyerSpecificCapability⟩. The exception occurs if the id field is * or wildcard. This indicates that the capability applies to all buyers. Using buyer-specific capabilities, suppliers can represent specific capabilities to certain buyers and generic capabilities applying to all other buyers. By not including a wildcard ⟨buyerSpecificCapability⟩ and only listing ⟨buyerSpecificCapability⟩s applicable to specific buyers, sellers can also represent the fact that they will trade only with a subset of all buyers. In cases where both the wildcard ⟨buyerSpecificCapability⟩ and a ⟨buyerSpecificCapability⟩ applicable to a specific buyer apply, the most specific ⟨buyerSpecificCapability⟩ is selected.

A schematic of a ⟨sellerSpecificPreference⟩ is given in Figure 2.

We begin at the top level of a capability:

$$\text{capability} \rightarrow \{\langle \text{buyerSpecificCapability} \rangle\}$$

where

⟨buyerSpecificCapability⟩ → id: ⟨id⟩,
discrete: {⟨discreteVarDescription⟩},
continuous: {⟨continuousVarDescription⟩},
range: {⟨rangeVarDescription⟩},
[discreteConstraint: ⟨discreteConstraint⟩],
instance: {⟨discreteCapabilityInstance⟩}
[aggregationParticipation: 0|1].

⟨id⟩ identifies a buyer or group of buyers. Individual buyers are represented by some unique identifier (say an integer) and the group of all buyers is identified by the wildcard character '*'. So we have

$$\langle \text{id} \rangle \rightarrow \langle \text{integer} \rangle \mid *\,.$$

aggregationParticipation is a Boolean flag giving the supplier's willingness to participate in

aggregate orders to the identified buyer.

Each of the variable constituent components is described by

$$\begin{aligned}
\langle \text{discreteVarDescription} \rangle &\rightarrow \text{name: } \langle \text{integer} \rangle, \\
&\qquad \text{allowedValues: } \{\langle \text{integer} \rangle\} \\
\langle \text{continuousVarDescription} \rangle &\rightarrow \text{name: } \langle \text{integer} \rangle, \\
&\qquad \text{min: } \langle \text{double} \rangle, \\
&\qquad \text{max: } \langle \text{double} \rangle \\
\langle \text{rangeVarDescription} \rangle &\rightarrow \text{name: } \langle \text{integer} \rangle..
\end{aligned}$$

In its simplest form, a ⟨discreteConstraint⟩ is a list of more primitive constraints

$$\langle \text{discreteConstraint} \rangle \rightarrow \{\langle \text{primitiveDiscreteConstraint} \rangle\}$$

where each primitive constraint is composed as follows:

$$\begin{aligned}
\langle \text{primitiveDiscreteConstraint} \rangle \rightarrow\ & \text{name: } \langle \text{string} \rangle \\
& \text{variables: } \{\langle \text{discreteVarName} \rangle\}, \\
& \text{includes: } 0 \mid 1, \\
& \text{values: } \langle \text{integerMatrix} \rangle
\end{aligned}$$

⟨discreteVarName⟩ is the name of the discrete variable involved in the constraint

$$\langle \text{discreteVarName} \rangle \rightarrow \langle \text{integer} \rangle.$$

The includes field is a bit. If the bit is 1, then the combinations listed in the values field are the allowed values the variables may take on. If the bit is 0, then the combinations listed in values are the excluded combinations, i.e., everything in the powerset of the variables is allowed except those combinations listed in values. The order of the variable names is significant, since they will be assumed to be in the same order in values. If there are a variables involved in the constraint, and c constraints, then ⟨integerMatrix⟩ is an $a \times c$ matrix of integers:

$$\begin{aligned}
\langle \text{integerMatrix} \rangle &\rightarrow \langle \text{integerVector} \rangle, \cdots, \langle \text{integerVector} \rangle \\
\langle \text{integerVector} \rangle &\rightarrow \langle \text{integer} \rangle, \cdots, \langle \text{integer} \rangle
\end{aligned}$$

The ⟨discreteCapabilityInstance⟩ component is described by

$$\begin{aligned}
\langle \text{discreteCapabilityInstance} \rangle \rightarrow\ & \text{mask: } \langle \text{discreteVarMask} \rangle, \\
& [\text{rangeCapability: } \{\langle \text{rangeVarInstance} \rangle\}], \\
& \text{continuousCapability: } \langle \text{continuousCapability} \rangle \\
& \text{continuousConstraints: } \langle \text{continuousConstraints} \rangle
\end{aligned}$$

A ⟨rangeVarInstance⟩ defines a range variable and has the form

⟨rangeVarInstance⟩ → name: ⟨integer⟩,
min: ⟨double⟩,
max: ⟨double⟩.

The ⟨discreteVarMask⟩ relates to the discussion of 6.2.2. As in Table 3 we have

⟨discreteVarMask⟩ → {⟨extendedVarValue⟩}

where an ⟨extendedVarValue⟩ is either an integer from the domain of the discrete variable or the wildcard character '*':

⟨extendedVarValue⟩ → ⟨integer⟩ | ∗ .

⟨continuousConstraints⟩ describes the hard linear constraints for the continuous variables. Since these constraints may be either inequality or equality, we have

⟨continuousConstraints⟩ → [equality: ⟨linearConstraints⟩],
[inequality: ⟨linearConstraints⟩]

Both the equality and inequality constraints are expressed through a matrix which is $c \times n_c$ where c is the number of constraints, and a vector which is $c \times 1$. Consequently we have

⟨linearConstraints⟩ → matrix: ⟨doubleMatrix⟩,
vector: ⟨doubleVector⟩

A ⟨doubleMatrix⟩ is defined by

⟨doubleMatrix⟩ → ⟨doubleVector⟩, · · · , ⟨doubleVector⟩

and a ⟨doubleVector⟩ is just what the name suggests – a vector of doubles:

⟨doubleVector⟩ → ⟨double⟩, · · · , ⟨double⟩.

The only remaining undescribed element above is ⟨continuousCapability⟩ whose description is

⟨continuousCapability⟩ → breakPoints: ⟨doubleListMatrix⟩,
valAtBreakPoints: ⟨doubleListMatrix⟩

⟨doubleListMatrix⟩ describes a $n_c \times n_c$ matrix whose elements are lists of ⟨double⟩:

$$\langle \text{doubleListMatrix} \rangle \rightarrow \langle \text{doubleListVector} \rangle, \cdots, \langle \text{doubleListVector} \rangle$$
$$\langle \text{doubleListVector} \rangle \rightarrow \langle \text{doubleList} \rangle, \cdots, \langle \text{doubleList} \rangle$$
$$\langle \text{doubleList} \rangle \rightarrow \{\langle \text{double} \rangle\}$$

It is assumed that the rows and columns of the matrix are in some canonical order so that we know which continuous variable is referenced. A natural order is the one defined in {⟨continuousVarDescription⟩}

Preferences

Just as capabilities may be buyer-specific so too may preferences be seller-specific. The same rules determining which seller-specific preference to apply are followed. A schematic of a ⟨sellerSpecificPreference⟩ is given in Figure 3.

We define a preference as follows

$$\langle \text{preference} \rangle \rightarrow \{\langle \text{sellerSpecificPreference} \rangle\}[, \langle \text{aggregatedPreference} \rangle]$$

i.e., a preference is a list of ⟨sellerSpecificPreference⟩ with an optional aggregated preference. We first describe ⟨sellerSpecificPreference⟩ and then consider ⟨aggregatedPreference⟩.

The ⟨sellerSpecificPreference⟩ is composed as follows

⟨sellerSpecificPreference⟩ →id: ⟨id⟩,
discrete: {⟨discreteVarDescription⟩},
continuous: {⟨continuousVarDescription⟩},
range: {⟨rangeVarDescription⟩},
dimensionWeights: ⟨dimensionWeights⟩,
discreteTradeoff: ⟨tradeoffTables⟩
[discreteConstraint: ⟨discreteConstraint⟩],
instance: {⟨discretePreferenceInstance⟩}

Of these elements, only ⟨dimensionWeights⟩, ⟨tradeoffTables⟩, and ⟨discretePreferenceInstance⟩ have yet to be defined. ⟨dimensionWeights⟩ gives the weights of all dimensions that indicate their importance. For convenience we break up the weights according to the three types of variables. Thus we have

⟨dimensionWeights⟩ → range: ⟨doubleVector⟩,
discrete: ⟨doubleVector⟩,
continuous: ⟨doubleVector⟩

A ⟨doubleVector⟩ has been described previously. Each of the corresponding vectors is as long

as the number of range, discrete, or continuous dimensions. ⟨tradeoffTables⟩ is an $n_{\text{discrete}} \times n_{\text{discrete}}$ matrix of ⟨tradeoffTable⟩:

⟨tradeoffTables⟩ → ⟨tradeoffTableMatrix⟩
⟨tradeoffTableMatrix⟩ → ⟨tradeoffTableVector⟩, ··· , ⟨tradeoffTableVector⟩
⟨tradeoffTableVector⟩ → ⟨tradeoffTable⟩, ··· , ⟨tradeoffTable⟩

A ⟨tradeoffTable⟩ is simply a matrix of double values.

Finally, we turn to the last undefined component of a ⟨preference⟩. A ⟨discretePreferenceInstance⟩ is composed as follows:

⟨discretePreferenceInstance⟩ → mask: ⟨mask⟩,
[rangeIdeal: {⟨rangeVarInstance⟩}],
continuousIdeal: ⟨doubleVector⟩,
tradeoffMatrix: ⟨doubleMatrix⟩,
[continuousConstraints: ⟨continuousConstraints⟩]

The rangeIdeal and continuousIdeal fields give the desired range and continuous trade parameters. The tradeoffMatrix field gives the positive definite matrix expressing the tradeoffs amongst the continuous variables. ⟨continuousConstraints⟩ have been described previously in the sell-side specification.

To complete the specification of preferences, we conclude with the definition of ⟨aggregatedPreference⟩. Refer to the discussion of section 6.1.7 for details.

⟨aggregatedPreference⟩ → participants: {⟨aggSpecification⟩},
contributionType: ⟨contributionTypeVector⟩,
additionalConstraints: ⟨continuousConstraints⟩,
discrete: {⟨discreteVarDescription⟩},
continuous: {⟨continuousVarDescription⟩},
range: {⟨rangeVarDescription⟩},
dimensionWeights: ⟨dimensionWeights⟩,
discreteTradeoff: ⟨tradeoffTables⟩
[discreteConstraint: ⟨discreteConstraint⟩],
instance: {⟨discretePreferenceInstance⟩}

In the above definition, the previously defined elements maintain their meaning. The additionalConstraints field allows the buyer to express constraints around the aggregation, such as "all orders must arrive on the same day," etc. participants lists the suppliers who can participate in the aggregation and their characteristics. Note that if the wildcard supplier participates, the order can potentially be aggregated across all suppliers. ⟨aggSpecification⟩

discrete mask	output	specificity
$[* \quad * \quad *]$	{continuous1, range1}	0
$[\varkappa_1 \quad * \quad *]$	{continuous2, range2}	1
$[\varkappa_1 \quad * \quad \varkappa_3]$	{continuous3, range3}	2

Table 3: Example of discrete masks for specifying continuous and range variables which are dependent on discrete variables. $\varkappa_1$ and $\varkappa_3$ signify specific values for the first and third discrete variables. The specificity of each mask is indicated in the third column.

describes information specific to a supplier participating in the aggregation. It is defined by

$$\langle \text{aggSpecification} \rangle \rightarrow \text{id: } \langle \text{id} \rangle.$$

id identifies the participating supplier and constraints specific to that supplier defined in an accompanying ⟨sellerSpecificPreference⟩ will be used in the optimization. Additional information may be added as required. The contributionType field is used to define the $\boldsymbol{\xi}$ vectors used in aggregation. The ⟨contributionTypeVector⟩ consists of n_c elements indicating the type of aggregation for each continuous dimension:

$$\langle \text{contributionTypeVector} \rangle \rightarrow \langle \text{contributionType} \rangle, \cdots, \langle \text{contributionType} \rangle.$$

Possible contribution types include

$$\langle \text{contributionType} \rangle \rightarrow \text{sum, average, zero.}$$

sum sets $\boldsymbol{\xi} = \mathbf{1}$, average sets $\boldsymbol{\xi} = \mathbf{1}/|\mathcal{K}(\boldsymbol{\varkappa})|$, and zero sets $\boldsymbol{\xi} = \mathbf{0}$.

Masking

We have allowed constraints, ideal values, tradeoffs, and continuous capabilities to be dependent on discrete variables. In this section we describe an efficient manner in which to encode this dependence.

The data structure must represent continuous and range variables for all valid discrete inputs. An efficient way to do this is to use hierarchical definitions. At the top of the hierarchy are the definitions of the continuous and range variables for the discrete values $\boldsymbol{\varkappa}^t = [*, \cdots, *]$. These values apply to all $\mathbf{v}$ unless more specialized masks are defined. A more specialized mask of the continuous and range variables is specified by defining values for some of the components $\varkappa_i$. The more components that are defined, the more specialized the definition. The most specific mask is always used. An example definition for three discrete variables is given in Table 3. The response to the lookup $[\tilde{\varkappa}_1 \ \tilde{\varkappa}_2 \ \tilde{\varkappa}_3]$ is {continuous3, range3} if and only if $\varkappa_1 = \tilde{\varkappa}_1 \wedge \varkappa_3 = \tilde{\varkappa}_3$, {continuous2, range2} if and only if $\varkappa_1 = \tilde{\varkappa}_1 \wedge \varkappa_3 \neq \tilde{\varkappa}_3$, and {continuous1, range1} otherwise.

Match Results

Returned to the buyer is a list of matches with different suppliers, which can be ranked and

viewed in many different ways in the GUI. A ⟨matchResultList⟩ is a list of matchResult:

$$\langle \text{matchResultList} \rangle \rightarrow \{\langle \text{matchResult} \rangle\}.$$

A match result may either be a ⟨singleSupplierMatchResult⟩ or an ⟨aggregatedMatchResult⟩:

$$\langle \text{matchResult} \rangle \rightarrow \langle \text{singleSupplierMatchResult} \rangle | \langle \text{aggregatedMatchResult} \rangle.$$

A ⟨singleSupplierMatchResult⟩ represents the best trade with a single supplier and is composed of the following elements:

⟨singleSupplierMatchResult⟩ → supplierId: ⟨integer⟩,
utility: ⟨double⟩,
feasible: 0|1,
[costFactors: {⟨double⟩}],
continuous: {⟨double⟩},
discrete: {⟨discreteVarDescription⟩},
range: ⟨rangeVarInstance⟩.

The supplierId indicates the supplier sourcing this trade and the utility field indicates the utility of the trade (which can be used to rank the trades). feasible is a bit indicating whether or not a feasible trade with this supplier was found. The continuous, discrete, and range fields list the respective trade parameters determined by the matching algorithm. The optional cost factors field lists the constituent costs contributing to the total cost of ownership C_o evaluated at the trade point returned in the ⟨singleSupplierMatchResult⟩.

An ⟨aggregatedMatchResult⟩ returns the optimal trade when the buyer has requested aggregation. It is composed of the following elements:

⟨aggregatedMatchResult⟩ → utility: ⟨double⟩,
feasible: 0|1,
[costFactors: {⟨double⟩}],
supplierTradeParameters: {⟨supplierTradeParameters⟩}.

As before, the utility field gives the utility of the aggregate trade, and the feasibility flag indicates whether or not a feasible aggregate trade was found (there may not be if the constraints were too stringent). costFactors can also be returned in C_o is sufficiently general to handle aggregated trades. Finally, ⟨supplierTradeParameters⟩ lists the trade parameters for each

supplier involved in the aggregation. It is defined as follows:

⟨supplierTradeParameters⟩ → supplierId ⟨*integer*⟩,
continuous: {⟨double⟩},
discrete: {⟨discreteVarDescription⟩},
range: ⟨rangeVarInstance⟩.

6.3 Summary

We have described an efficient computational procedure in which to encode buyer's trading preferences and hard constraints, supplier's delivery capabilities and constraints, and optimize to find those matches between one buyer and one or many sellers that maximize the buyer's utility. By optimizing against both qualitative and quantitative factors, and exploiting the trading flexibilities possessed by both parties, the system determines better trades. The tool is particularly useful as companies move their direct material purchasing online. By optimizing across flexibilities, win-win trades are discovered for both trading parties.

The representation of trading preferences is designed to be expressive yet easily elicitable from a buyer, and computationally tractable. The representation of supplier capabilities was chosen to parallel the manner in which suppliers already think of their delivery capabilities and seamlessly includes volume discounts and incremental costs. These supplier capabilities may be part of an online catalog. The representation of the negotiation space is rich, supporting three types of variables.

We have outlined a manner in which preferences may be inferred automatically through models of the purchasing company. Such models incorporate many cost factors, taking the total cost of ownership into account. The system provides trades which minimize the total cost and represent significant new savings.

The invention can operate both at a buyer's site, where suppliers input their capabilities through an HTML interface to the world wide web or as an embedded part of an electronic market hosted by a particular web site. The invention may operate at regularly scheduled intervals or sporadically in lieu of current request for quotations (RFQ). The buyer may broadcast a RFQ event to suppliers, indicating a time within which suppliers must respond. At the close of the event, the buyer can use the present invention to assist in the analysis of the supplier responses.

Complex algorithms have been specified which should permit most matching optimization to occur in near real-time. The rapidity of optimization, combined with graphical what-if tools, allows for analysis and exploration of trades, which should significantly improve the quality of purchasing decisions.

6.4 An e-Commerce Infrastructure for Value Chains

In this section we describe in detail how the proposed infrastructure delivers on the promises made in the Summary of the Invention. We begin by describing major innovations in the present invention and how they are all used synergistically.

6.4.1 Major Innovations

The most broad invention combines at least four advances:

1. multidimensional automated markets (hereafter simply markets) which capture many aspects of value.

2. algorithms and interfaces which implicitly allow for consumers to express the preferences over the multiple dimensions of value

3. linked markets allowing for complex assembly of products

4. specification of the constraints (both logical and numerical) inherent between markets to allow for coordinated buys and sells between markets

In addition, inventions described in a patent application titled, "An Adaptive and Reliable System and Method for Operations Management", Application Number 09/345,441 filed July 1, 1999 (the contents of which are herein incorporated by reference)can also be used in conjunction with the present invention. These other inventions are:

5. the use of models and optimization algorithms to optimally determine the best bids to submit to the automated market

6. the use of subset relations `is-a`, `has-a` *etc.* to automatically construct the constraints between markets

6.4.2 Integrated Picture

The internet and e-commerce are changing the way consumers and businesses trade with each other. One interesting trend is the development of centralized economic hubs (*e.g.* eSteel, ChemDex) through which all e-commerce in a particular domain flows. This trend is expected to continue and become more prevalent. The invention described here is the tool that drives these economic hubs.

Before describing the components and innovations in detail we describe the overall architecture of the integrated system. For concreteness we focus on the invention as applied to the personal computer (PC) supply chain. We note that the ideas can be applied equally well to supply chains in other industries.

The infrastructure is composed of three major components all running on different computers. A central server (or more likely servers) coupled to the economic hub serves as the

source of consumer demand. Other sets of servers act as the trading markets themselves relaying information between buyers and sellers. Finally, running at each market participant's site is client software coupled to the relevant markets.

Consumer Demand: Consumer demand is pulled through the supply chain through a set of coupled reverse auction[17] markets. An interface iteratively elicits multi-dimensional consumer preferences which is then forwarded to a top level market where suppliers offer to fulfill the consumer request. The functioning of the consumer interface is described in more detail in section 6.4.3.

Markets: Markets within the supply chain are represented by servers which link trading partners. The market broadcasts demand, collects responses, and forwards results back to the source of the demand. See section 6.4.4 for more information on the functioning of markets, particularly the coupling between markets.

Client Companies: Each trading participant (whether buying or selling) is represented by client software connected to the appropriate markets. The client software both initiates requests and responds to requests from other market participants. The client software running at the company also maintains a memory of all past transactions and can eliminate those that are out of date. A user interface allows companies to define the operation of their interface to the markets. See 6.4.5 for further details.

Supplier Push As has been mentioned the infrastructure is organized around demand pulled through the supply chain by the consumer. Interestingly, the same pull infrastructure can also be used by suppliers to push demand. In addition to requesting and responding to requests, the market interface running locally at company sites could also be used to advertise capability (capability not in response to any particular demand) to the market which will then forward the advertised capability to any participant connected to the market.

6.4.3 Climbing in Preference Space: the consumer interface

The consumer drives the integrated infrastructure by generating demand. In our example, a graphical user interface (GUI) at the PC economic hub provides a centralized interface through which all computers are purchased. The GUI provides a number of advantages to both consumers and computer vendors. A computer is described across multiple dimensions of value. Some of these dimensions include: price, memory size, processor speed, hard drive side, removable storage, monitor size and resolution, financing, warranty, delivery date, *etc.* For each dimension the user can identify a preference. For example the consumer may wish 128Mb of memory, a 600MHz Pentium III processor with a 15Gb hard drive for $1500 delivered within 3 weeks. Additionally, it may be important for the consumer to express constraints that must be satisfied. To keep things as simple as possible we express an optional inequality constraint for each dimension, *e.g.* price must be less than $1600 and the hard drive must be at least 12Gb.

[17] A reverse auction is like an auction except that a request is broadcast to suppliers who bid on fulfilling the request. Unlike traditional auctions the proposed market clears in one shot, *i.e.* each supplier has only one chance to bid.

Optional Dimensions: This example brings up an interesting feature of the dimensions of value. Imagine for a moment that one of the dimensions was portability with possible values being high portability, medium portability, and low portability. In each of these cases we might consider lightweight laptops, heavy laptops, and desktops. Any one of these choices might then invoke additional situational dimensions. For example, if we select high portability then battery life (in hours) becomes an important dimension which is only applicable to laptops. There are a number of ways to treat these optional dimensions. The simplest possibility is to have separate markets for laptops and desktops with appropriate dimensions for each. A better solution is to have a GUI which only turns on the optional dimensions when appropriate. If low portability is selected by the user the battery life dimension remains lightly greyed out and inaccessible. If medium or high portability is required then the battery life dimension is activated. In the technical details section we show how this can easily be accomplished.

Dimension Weighting: Optionally, the user may also specify an importance to each dimension. If hard drive size and price are paramount these two dimensions have high value while all other dimensions have low value. The weighting is used to identify similarity between computers; two computers which are quite different only in one dimension which is not highly weighted by the consumer are considered quite similar. See the technical details section.

Threshold Constraints: A final way in which consumers may specify their needs is through constraints expressed for each dimension. The constraints are simple to state. The consumer specifies a threshold and whether or not he wants to be above or below that threshold. For example the consumer may specify that the monitor size has to be at least 17 inches and he will not pay any more than $1500. Knowledgeable consumers may define all these settings directly or the software may infer settings for less experienced consumers based upon simple questions (like the major uses of the computer).

Branding: Consumers may also have brand preferences for a computer or any of its components. The proposed GUI allows for users to express these preferences. There are two alternative ways in which branding might work and depending on the application either may be appropriate. In the first alternative brand name is a hard constraint and no computers are shown to the consumer unless the brand name matches the consumers desire. In the second alternative brand name is a soft constraint and the consumer merely prefers one brand over another.

In either case the GUI allows the user to specify for each dimension whether or not brand matters and if so which brand/brands is/are preferred.[18] If brand is a hard constraint this constraint is propagated to the top level market and all suppliers of the top level market so that no responses not respecting the constraint are ever returned to the consumer. If brand is a soft constraint it is incorporated into the distance function d_i as described in the technical details section. Otherwise identical components which match the desired brand are closer than those components which do not match the desired brand.

Iterative Hillclimbing and Recombination: Once the consumer has specified a desired com-

[18] We might also allow consumers to rank their brand preferences but this extension seems unnecessary in light of the iterative nature of product exploration.

puter defined as a point (and perhaps weights and constraints) in the high dimensional value space, a query is sent to a top-level automated market. Sellers of computers (*e.g.* Micron, Gateway, Dell) attached to the market respond to the consumer request by returning offers (one or many) to the market for candidate computers again represented in the high-dimensional space of value. Responses to the consumer are filtered to satisfy the specified constraints. The GUI allows the consumer to sort responses based on any of the dimensions (see the d_i function in the technical details section) or according to a global measure of distance between two computers (see the d function in the technical details section).

The consumer may modify any of the responses (a mutation) and use this as a resubmission to the market. This method of exploring the space of possible computers according to the consumer's desires and the supplier's availability can broadly be summarized as climbing in preference space. The climbing can begin from the most recently visited computer configuration or from any configuration that has been saved in a history list.[19] It is also perfectly feasible to allow the consumer to recombine desirable computer configurations. For example the user may ask for configurations which are "in between" two previously examined configurations and combine the advantages of both configurations. We also might allow the consumer to submit more than a single configuration to the top level market and receive responses around all submissions.

Benefits of the Approach: This procedure offers a significant benefit over other approaches to multidimensional markets. In systems like OptiMark[20] the consumer (in this case a financial trader) must a priori specify preferences over all dimensions (price and quantity). The explicit specification of preference over even two dimensions is a difficult task and it is unreasonable to expect consumers to be able to do this. The present innovation allows consumers to iteratively explore preference space thereby implicitly defining preference in a friendly manner.

This method of interactive consumer exploration is suitable for any complex product and we expect the application to be broad. The present invention has many other applications[21].

Technical Details We have mentioned that computer configurations can be sorted on all dimensions according to how closely they match the consumers preferences. In this section we describe how this sorting might work and how it interacts with the weights and brand preferences attached to each dimension.

The value space defines a notion of distance or similarity between products (in this case computers) defined as follows: If there are a total of D dimensions[22] of value then a computer $\mathbf{c}$ is described as a D-vector of the settings for each dimension. Let i label the components of $\mathbf{c}$ so that for example c_1 = memory is c_2 = processor speed *etc.* For example, the memory dimension might assume any of the values c_1 = 64Mb, c_1 = 128Mb, or c_1 = 256Mb and similarly for all other dimensions. Let a_i be a binary variable (*i.e.* a_i takes the value 0 or 1)

[19] History lists extend both within and between sessions.

[20] U.S. Patent 5,845,266, "Crossing network utilizing satisfaction density profile with price discovery features" dated 12/1/98.

[21] *e.g.* credit cards where the dimensions might be interest rate, yearly fee, grace period, benefits (like frequent flyer miles, *etc.*)

[22] The D dimensions include both required and optional dimensions.

indicating which dimensions are actually used. If c_j is the dimension corresponding to battery life in hours then $a_j = 1$ indicates this dimension is appropriate while $a_j = 0$ indicates the dimension is not appropriate. The vector whose components are the a_i is represented by $\mathbf{a}$.

A consumer preference also includes optional weights and preferred brand. The weight (or importance) of dimension i is indicated by w_i. All w_i are non-negative and normalized so that $\sum_{i=1}^{D} a_i w_i = 1$. The vector of all weights is represented by $\mathbf{w}$. The brand preferences for dimension i are expressed as a (possibly empty) set $\mathcal{B}_i$ which includes desired brands. $\boldsymbol{\mathcal{B}}$ is the set of all $\mathcal{B}_i$. A complete consumer description is thus represented by the vector $\boldsymbol{\mathcal{C}} = \{\mathbf{c}, \mathbf{a}, \mathbf{w}, \boldsymbol{\mathcal{B}}\}$ with components $\mathcal{C}_i = \{c_i, a_i, w_i, \mathcal{B}_i\}$.

The distance $d(\mathbf{c}, \mathbf{c}')$ between computers $\mathbf{c}$ and $\mathbf{c}'$ is then defined as

$$d(\boldsymbol{\mathcal{C}}, \boldsymbol{\mathcal{C}}') = \frac{\sum_{i=1}^{D} a_i w_i \, d_i(\mathcal{C}_i, \mathcal{C}_i')}{\sum_{i=1}^{D} a_i} \tag{8}$$

where $d_i(\mathcal{C}_i, \mathcal{C}_i')$ measures the distance along the single dimension i. To allow for standard comparison between computer configurations we normalize all d_i so that they lie between 0 and 1.

We recall from previous discussion that for soft brand constraints the matching of a brand to its desired value is incorporated into the distance function. Thus we generally write the distance function d_i as the sum of two terms, one for brand and one for similarity. If b_i is an indicator variable whose value is 1 if the consumer has identified that brand matters for the ith dimension (*i.e.* $\mathcal{B}_i$ is non-empty) and 0 otherwise, then we write

$$d_i(\mathcal{C}_i, \mathcal{C}_i') = \alpha_i b_i d_i^{(b)}(\mathcal{B}_i, \mathcal{B}_i') + (1 - \alpha_i b_i) d_i^{(s)}(c_i, c_i'). \tag{9}$$

The α_i parameter determines the relative importance between brand name and similarity. Lacking better information we might take all $\alpha_i = 1/2$.[23]

For simplicity we assume that the brand distance function, $d_i^{(b)}(c_i, c_i')$, is 0 if there is overlap in the preferred brands for c_i and c_i' and 1 otherwise, *i.e.*

$$d_i^{(b)}(\mathcal{B}_i, \mathcal{B}_i') = \begin{cases} 1 & \text{if } \mathcal{B}_i \cap \mathcal{B}_i' = \emptyset \\ 0 & \text{otherwise} \end{cases} . \tag{10}$$

The similarity distance measure, $d_i^{(s)}(c_i, c_i')$, is only slightly more complicated. The allowed values for most dimensions are discrete over a finite range. If the ith dimension is bounded and ranges from $c_i^{\min}$ to $c_i^{\max}$ then we might define

$$d_i^{(s)}(c_i, c_i') \equiv \frac{|c_i - c_i'|}{c_i^{\max} - c_i^{\min}} \tag{11}$$

though other choices are certainly possible and may be more appropriate depending on the nature of the dimension. In many situations it is appropriate that the distance function is not

[23] Better values for α_i might be identified from real consumer interaction with the GUI.

symmetric, *i.e.* $d_i^{(s)}(c_i, c_i') \neq d_i^{(s)}(c_i', c_i)$. Examples of asymmetry are easy to find. A consumer may be more than willing to accept an extra few Mb of memory but will be very dissatisfied receiving a few Mb less. As a concrete example we may offer open ended ranges on most choices. The range is defined by a single point x_i' and an indication of whether the range is above or below that point. In the case of the range $[x_i', \infty)$ we can take the distance function measuring the distance from the range to be $d(x_i, x_i') = (x_i' - x_i)\theta(x_i' - x_i)$. In the opposite case where the range is $[0, x_i']$ the distance function can be $d(x_i, x_i') = (x_i - x_i')\theta(x_i - x_i')$. Accordingly, the present invention also includes possibilities for $d_i^{(s)}(c_i, c_i')$.

Distance Ranking: The distance metric of Eq. (8) allows for a global ranking (after configurations not satisfying the constraints have been filtered out) of vendor responses based upon their similarity to the original consumer request (smaller distances are more similar). If $\mathcal{C}_{\text{req}}$ describes the consumer specified computer and α labels responses from vendors in response to the request then the automated market sorts the responses based upon $d(\mathcal{C}_{\text{req}}, \mathcal{C}_\alpha)$ and returns a specified number back to the consumer interface. The interface also allows sorting (in order of decreasing distance) on each dimension according to the d_i. The consumer then iterates the process starting from any of the returned offers, modifying them slightly and re-submitting to the market. In this way the consumer can move about in the space of preferences climbing towards ever more desirable computer configurations. When the consumer finally does identify a configuration that he desires to purchase it is a simple matter of hitting a buy button.

6.4.4 Integrating the Supply Chain: cascading markets

It is natural to consider the extension of high-dimensional automated markets to other transactions made in the supply chain. In fact, this may be essential to fulfill the kind of real-time consumer preference exploration described in section 6.4.3. These other markets will also typically be high dimensional but the dimensions will differ from the top-level market. For example there may be a hard drive market where buyers of hard drives (whether computer vendors, consumers, or value added retailers) meet sellers (*e.g.* suppliers like Seagate, Western Digital). The dimensions of the hard drive market might include: price, size (in Gb), type (IDE vs SCSI), quantity, delivery date, *etc.*

Operation of Cascading Markets: The basic idea is straightforward. Before a computer vendor responds to consumer requests in the top-level market the vendor may send its own requests to submarkets to purchase products and services it needs to meet the consumer demand. Suppliers to the computer vendors may in turn go to other submarkets before responding to computer vendor requests. A single consumer request may then result in real-time cascading inquiries to a tree of nested markets all the way down to the beginning of the supply chain. We call the market which originated a request the parent market and the submarket a child market. It is a requirement that any possible path through the cascading markets is acyclic.

When the consumer moves on to examine a new configuration all relevant market participants (*i.e.* those that responded to the initial customer request or any concommittant

subsequent request) are informed and their offers are released. If, on the other hand, the consumer elects to purchase the configuration all participants are informed and are held to their offers. When combined with emerging XML standards for e-commerce (*e.g.* RosettaNet in the IT industry) the necessary "paperwork" for confirmed purchases can be generated by the computer and sent between all relevant parties (including the consumer after a credit card check).

Note that it remains unlikely that every consumer purchase will trigger a cascade of transactions throughout the tree of markets because it is not profitable in most supply chains to order and transport in single unit batches. Thus, only when inventories need to be replenished will companies in the supply chain go to the automated market in response to a request.

Auxiliary Information: It is useful to pass additional information between markets beyond merely the parameters describing the trade. There are many reasons extra information may be desired and we describe two applications of auxiliary information. Most simply, in cases where it is important to identify who the buyer or seller actually are an additional identifier may be passed labeling each trader.

A more serious application addresses a potential problem of nested markets. A problem with cascading markets is that demand can spuriously be amplified as it moves between markets and away from its originating source. Consider a consumer demand for 100 computers. If two of the computer suppliers are both low on hard drives they might both submit a request to the hard drive market for an additional 100 drives. If the hard drive company were to estimate demand based solely upon the hard drive market then in this case the company would falsely assume that the demand was twice as high as it really is. This problem is easily fixed by supplying auxiliary information along with trade parameters. A request that comes into a toplevel market is assigned an identifier indicating the market and a unique identifier.[24] Every request to submarkets that is in response to a request in the parent market is again assigned both market and trade identifiers. These identifiers are appended to the identifiers from the parent market when the request is broadcast to all suppliers listening to the market. Returning to the hard drive example, if the top-level market has market identifier ASSEMBLE and the hard drive market has identifier HARDDRIVE and if the original consumer request had trade identifier 4561133 and the market trade identifiers were 56891 and 56892 respectively then the identifiers which are passed to all hard drive suppliers in response to both hard drive demands are ASSEMBLE:4561133,HARDDRIVE:56891 and ASSEMBLE:4561133,HARDDRIVE:56892 respectively. In this way all hard drive suppliers can see that the request originated with the same demand. In fact such mechanisms might also allow for improved forecasting of demand since all demand is accompanied by its path through the entire supply chain which may contain additional useful information.

Expiration Dates: One of the advantages of the present e-commerce infrastructure is the real-time setting of trade parameters. Companies can use the system to optimize their operations continuously. To achieve the maximal benefit from continuous optimization it is important that the offers made by any market participant have a bounded lifetime. What

[24] All trades in that market will have distinct identifiers.

is optimal now may not be in a day (or even an hour). Thus an additional application of auxiliary information is the inclusion of expiration dates after which the trader parameters are invalid. Unless perishable trades are executed before the expiration date they are removed from the system.

Other B2B Market Issues The cascading submarkets are all business to business markets and as such can operate under different rules than the business to consumer market. In this section we address what some of these differences might be.

Clearing Mechanisms: We have proposed that the B2B markets clear in a fashion similar to the B2C market. One important difference is that these markets probably must clear in one shot in order to deliver real-time feedback to the end consumer. There are certainly other ways in which the B2B markets clear. One possibility is to use optimization tools to define satisfaction surfaces just as in OptiMark. In this way the parameters of the trade are determined by software to optimize some joint criteria. Ideally, we would like protection for other B2B market clearing mechanisms.

Aggregation: Another potentially important function of automated markets is the aggregation of orders to meet demand. For large orders it may be the case that no single supplier can meet the requested demand. Market software can automatically collect responses to fulfill orders. This software, in addition to coordinating between purchases in different markets (see section 6.4.5) can also be used to coordinate multiple orders within the same market.

Many-to-many markets: Thus far the market mechanisms we have described all result in one-to-one trades (except if we aggregate orders in which case a trade may be many-to-one). For future reference we also note that the markets themselves may be more complex and used for many-to-many trades.

6.4.5 Coordination Between Markets: inter-market language

The assembly of a product or service from constituent products or services often requires coordinated purchases from a number of supply markets.

Logical Constraints: As illustration, there is no point in ordering a part if transportation can not also be arranged at the appropriate time. Both events must occur or else neither of the purchases should be made. This is an example of a Boolean logical constraint (in this case the logical **and** function) that exists between multiple markets. Of course in more complicated situations there may be more complex logical constraints between markets.

For added expressivity we propose using linear logic as the language in which to express logical constraints. Linear logic offers compelling advantages over Boolean logic since it subsumes Boolean logic and is resource sensitive exactly as real supply chain transactions are. Efficient implementations of linear logic exist so that checking of linear logical constraints is straightforward. The present invention includes support for both Boolean and linear logical constraints.

Numerical Constraints: There is another important class of relationships that may exist between markets. In most complex processes certain events have to occur sequentially; the

process can not progress until some condition is fulfilled. Our shipping example also provides an example of this. There is no point arranging the purchase of transportation if the only available pick up date is before the item to be shipped can be completed. This is an example of a numerical constraint that exists between the parameters of transactions on different markets and optionally some description of the state of the company. Other examples might include cost constraints which require that the total expenditures on all purchases to be less than some threshold.

A Constraint Language: Any automated market solution to supply chain coordination problems will require these constraint mechanisms and generically these constraints will be either logical (as in the **and** constraint) or numerical (as in the cost constraint). We propose an Inter-Market Language (IML) in which to express these constraints so that the nested market solution always respects the user specified constraints.

IML constraints are specified as follows. Let m_i be a logical variable which is true if a transaction is made on the ith market (and false otherwise) and let $m_i(\mathbf{x}_i)$ be a logical variable which is true if the transaction occurs with multidimensional parameters $\mathbf{x}_i$.[25] Furthermore let $\mathbf{s}$ be a vector of numerical values (for example describing current inventory levels, lead times, *etc*) and $\tilde{s}$ be a vector of logical values describing the state of the company. The two classes of IML constraints which both must simultaneously be satisfied are then expressed as

$$B(m_1, \cdots, m_M, \tilde{s}) \tag{12}$$

$$N_=(\mathbf{x}_1, \cdots, \mathbf{x}_M, \mathbf{s}) = 0, \quad N_<(\mathbf{x}_1, \cdots, \mathbf{x}_M, \mathbf{s}) < 0 \tag{13}$$

where $B(\cdot)$ is an arbitrary logical function, there are M relevant submarkets, and the functions $N_=(\cdot) = 0$ and $N_<(\cdot) < 0$ express the equality and inequality constraints respectively.

Operational Use of Constraints: Operationally the coordination amongst different market transactions is achieved as follows. Software representing the interests of a vendor (and therefore probably running at the vendors site) expressed in IML listens to servers representing various markets. When a market request that the vendor wishes to respond to is heard the software examines the companies internal state $\{\mathbf{s}, \tilde{\mathbf{s}}\}$ and determines what items, if any, need to be purchased on child submarkets. The software defines the purchase by defining a point or the desired parameters for each trade in each market. These requests are sent to servers representing each child market. The client software running at each company then waits for responses from each child market. If there are M total markets queried and n_i responses from the ith market then there are a total of $\prod_{i=1}^{M} n_i$ possible combinations of deals. The software then examines each deal filtering out those combinations which fail to satisfy the constraints of Eqs. (12) and (13). Thus we refer to this aspect of the client software as filtering. Those combinations which pass the constraints can then be ranked (either by a human user or by additional company specific criteria (see reliability below and in section 6.4.7) added to the client software running at the vendor site) and sent back as responses to the original request.

We have operationally described how constraints and filters may be used to coordinate

[25] m_i and $m_i(\mathbf{x}_i)$ are related by $m_i = \vee_{\mathbf{x}_i} m_i(\mathbf{x})$ where by $\vee_{\mathbf{x}_i}$ we mean the logical **or** of $m_i(\mathbf{x}_i)$ over all possible values of the parameters $\mathbf{x}_i$.

purchases from automated markets. The converse problem is the coordination of sell orders to automated markets. This is less likely to be problematic since usually there is only one sell order being responded to at any given time and thus there are no sell-side coordination problems. Nevertheless, if there are sell-side coordinating activities that must occur the same mechanism can be used. A separate set of logical and numerical constraints can be defined in IML for sell orders. Any responses to requests can then be checked against these constraints to ensure no responses are sent which violate the selling practices expressed through the IML constraints. Sell-side filters can be used to enforce preexisting contracts like specially arranged pricing for preferred customers. This may be particularly important when responses to requests are submitted automatically by software coupled to the companies ERP software. Typically such software will appropriately construct a response but when no human is present to check responses sell-side filters provide a final sanity check.

Other Applications of Constraints: An important consideration for any company in a supply chain is reliability. The company must be assured that its suppliers meet their obligations. In current supply chains this consideration is important enough that companies often limit themselves to a handful of trading partners with whom they maintain long term relationships. In order to fully reap the benefits of automated markets where many vendors may fulfill a need we need to resolve the issue of reliability. Filters are one way to address this issues. If for some historical reason a particular company doesn't like some suppliers the company may add these to be filtered. Thus the internal state $\{\mathbf{s}, \tilde{\mathbf{s}}\}$ may include references to unwanted suppliers. Alternatively, reliability may be used to rank bids that pass all filters.

6.4.6 Practical Concerns

In this section we show how real world concerns of scalability, real-time performance, reliability, and security can be assured within the proposed e-commerce framework.

Scalability The utility of an automated market is directly dependent on the liquidity (volume of trades) of the market. The greater the number of participants the greater the value to any individual participant. It is thus essential that the system scale gracefully to large numbers of market participants. Because the proposed e-commerce infrastructure is distributed scaling to large sizes is not a problem.

The market is driven by consumer demand pulled through the supply chain. Interfaces to consumer interaction can run locally on the consumer's computer as Java language applets. The central server (or servers) for consumer interaction is then only responsible for relaying consumer information (expressed in compact XML documents) into the top level market and returning the list of consumer responses to the appropriate user computer. The workload is shared across multiple machines with more machines involved the greater the number of consumers.

Similarly, the automated markets themselves serve mainly as relays of information and have little computational demands placed upon them. The markets send out requests, collect responses, and inform winning bidders of actual purchases. The bulk of the necessary

computation is in the software which determines the parameters of the bids and responses. Since this software runs locally on each company's computer the system naturally exploits the parallelism inherent with increased vendor participation.

Real-Time Response To be most useful to consumers it is important that consumer feedback is in real or near real time. This may be problematic on the rare occasions in which a query triggers a cascade down through submarkets. Before the top level market can clear (*i.e.* for responses to come back to the consumer) the tier one market must clear and before the tier one market can clear the tier two market must clear, *etc.* Fortunately, it is likely that all B2B markets will be monitored by software so that turnarounds on all markets will be fast. Market time-outs beyond which responses will not be accepted ensure timely responses from all submarkets.

Nevertheless, in situations where the response time is slow we can offer the consumer an immediate but approximate response and warn the consumer that the response may not be accurate. This can be accomplished by caching previous responses and estimating the response by interpolation of similar previous responses. Quick algorithms for the interpolation are easy to devise as long as the cached data can be accessed quickly as in any modern database.

6.4.7 Reliability

For any company to rely on the proposed infrastructure to run its supply chain it must be assured that the system is always up and running. For the system to be functioning for a company its own local server has to be up as well and all markets must be running. Each company is responsible for ensuring that its local servers are operating. The markets themselves will be served by a network of computers sharing the responsibility for managing transactions. The software which manages transactions can ensure that if a connection is broken during mid-transaction (*e.g.* if a computer goes down) it will be correctly reconstituted and resent. Such software is currently available from software companies.

6.4.8 Security

Whenever real dollars are being exchanged electronically there will always be important issues around security. We address each of these issues.

Commitment Assurance: It is important the every participant is confident that contra parties honor their commitments whether in terms of payment or delivery. We have described one mechanism that allows companies to automatically filter out those companies with whom they have had bad past experiences. It might also be expected that word of any party defaulting on its commitments would rapidly spread to all other participants making any default very costly for the offender.[26] If these mechanisms are insufficient the infrastructure may require each participant to pay an up front fee from which offenders compensate their contra partners.

[26]We might post defaults publicly so that other parties rapidly become aware of this knowledge and add the offender to their filters.

Authentification: The system must ensure that all parties are who they say they are. This is an old problem faced by internet participants and has been solved satisfactorily by systems like PGP *etc.*

Privacy: It is also important that companies private data remains private. If a competitor could intercept a companies responses to the market it would have a distinct and unfair advantage. All communications will be strongly encrypted to assure that this can not occur.

Information Visibility: Similarly we may want different companies to have differing access to information.[27] For example data about consumer demand may be obtainable by a participant but only at a fee. This information may be served based on the authentification of the participant.

6.4.9 Optimization of the Supply Chain

The final ingredient in the infrastructure has been alluded to but not explicitly discussed. To achieve maximal benefit from the flexibility the proposed invention introduces at all levels of the supply chain, real-time optimization systems can be used. This innovation is described in a patent application, titled, "An Adaptive and Reliable System and Method for Operations Management", Application Number 09/345,441 filed July 1, 1999 (the contents of which are herein incorporated by reference).

Coupling to ERP Software: The optimization systems monitor (in real-time) the state of the company and extrapolate from models of the company in its supply chain the optimal responses to requests that come in on the automated markets that the company supplies to. To achieve the best benefit, the servers or filters running at each company are coupled to ERP software to determine the best real-time response. The optimal response will be a function of the current instantaneous state of the company (as recorded in the ERP software) coupled with modeling and planning capabilities. A model of the likely future short term evolution of the state of the company may be used to best exploit the flexibility inherent in meeting the demand from the automated market. Such models might also take into account future expected demand and future responses from suppliers.

Another important optimization that might yield significant payoffs is in the tradeoff between making a response that is very close to meeting a requester's demand versus the savings obtained by perhaps being further from the requesters true desires.

6.4.10 Self-Organizing Supply Chains

In the procedure outlined above it is the responsibility of each company to define its business rules as expressed though its operating constraints and ranking criteria. This is not a unreasonable request and is a formalization of the company's current operating procedures. However, the present system does suggest a novel extension for the supply chain of the future. We can view the activities along a supply chain resulting in a finished good or service as an exercise in theorem proving in linear logic. Each company in the supply chain expresses its

[27] This will certainly be the case if we wish to charge for access to some of this information.

capabilities as predicates and functions (to include constraint) in linear logic. The finished good or service is a logical expression of a set of predicates. A path or method of assembly of the final good can then be represented as a proof of the logical expression representing the finished good or service. If the good can not be built then no such proof will be found. The precise manner in which the product is assembled is read off the proof net describing all the necessary steps. The beauty of this approach is its simplicity and the availability of theorem provers for linear logic. Though theorem proving for linear logic can be NP complete proofs of supply chains may be much simpler. To summarize: the mechanism of self-organization in the supply chain is *proving* the final good or service.

What are some of the axioms describing operations in the supply chain? There is shipping, manufacturing from components, etc. The state of the company includes inventory levels, lead times, etc.

6.4.11 Learning in the Infrastructure

There are many ways in which the data naturally recorded by the proposed infrastructure can be usefully used.

Prediction of customer demand: All the markets (both B2B and B2C) are rich sources of data for mining. Minimally, within any market we can use the data for both forecasting and inference of true customer desires (i.e. what they really want and now just what they are buying). The quantitative and nature of this market data should make mining it simpler than traditional data sources.

Inference of consumer preferences: The present invention also improves upon the efficiency of the end consumers search by aiding the search with learning tools. Based on the consumers interaction with the top-level market software can extrapolate to guess preferences and speed the search.

While the above invention has been described with reference to certain preferred embodiments, the scope of the present invention is not limited to these embodiments. One skilled in the art may find variations of these preferred embodiments which, nevertheless, fall within the spirit of the present invention, whose scope is defined by the claims set forth below.

8 References

[1] Keeney, Ralph L., Raiffa, Howard, *Decisions with Multiple Objectives*, Cambridge University Press:Cambridge, 1993.

[2] Saaty, Thomas L., *The Analytic Network Process*, RWS Publications:Pittsburg, 1996.

[3] A concise introduction to discrete choice analysis with a comparison to conjoint analysis is found at `http://www.action-research.com/discrete.htm`.

[4] Nesterov, Yurii, Nemirovskii, Arkadii, *Interior Point Polynomial Algorithms in Convex Programming*, Society for Industrial and Applied Math:Philadelphia, 1994.

[5] Tsang, Edward, *Foundations of Constraint Satisfaction*, Academic Press:London, 1993.

Petitpas Exhibit 2

9 Abstract

A system is described which allows buyers to define their preferences and sellers to define their capabilities, then determines which trading points maximize the utility of the buyer. The system suggests trades by exploiting the flexibilities and tradeoffs encoded by both parties, thus providing win-win trades. A second level of optimization ranks the trades with all suppliers, allowing the buyer to rapidly determine the best alternatives. The system allows for rich negotiation spaces and supports continuous, discrete, and range or interval decision factors.

Petitioner's Exhibit 2009

A Distance Function for Range Variables

The distance between buyer range variable $r_i = (\underline{r}_i, \overline{r}_i)$ and seller range variable $r_j = (\underline{r}_j, \overline{r}_j)$ is related to the overlap defined as

$$\text{overlap}(r_i, r_j) = \int_{\mu_j - \sigma_j}^{\mu_j + \sigma_j} dx\, \mathcal{N}(x; r_i)\mathcal{N}(x; r_j)$$

where $\mathcal{N}(x; r_i)$ is a Gaussian in x centered at $\mu_i = (\underline{r}_i + \overline{r}_i)/2$ with standard deviation $\sigma_i = \alpha(\overline{r}_i - \underline{r}_i)$ and α is a tunable parameter. The range of integration is bounded by the seller's capabilities. Evaluating the integral, we find

$$\text{overlap}(r_i, r_j) = \frac{1}{2\sqrt{2\pi(\sigma_i^2 + \sigma_j^2)}} \exp\left[-\frac{(\mu_i - \mu_j)^2}{2(\sigma_i^2 + \sigma_j^2)}\right] \left(\text{erf}\left[\frac{\sigma_j^2 + \sigma_j\mu_i - \sigma_j\mu_j + \sigma_i^2}{\sigma_i\sqrt{2(\sigma_i^2 + \sigma_j^2)}}\right] - \text{erf}\left[\frac{-\sigma_j^2 + \sigma_j\mu_i - \sigma_j\mu_j - \sigma_i^2}{\sigma_i\sqrt{2(\sigma_i^2 + \sigma_j^2)}}\right]\right).$$

By adjusting α we can control the hardness of the boundary at the ends of the range (smaller α corresponds to sharper dropoff outside the range). The maximum overlap will occur when $\mu_j = \mu_i$ and $\sigma_j = 0$. Substituting those values in the above equation, we get

$$\text{maxOverlap} = \frac{\text{erf}\left[\frac{1}{\sqrt{2}}\right]}{\sqrt{2\pi(\sigma_i^2 + \sigma_j^2)}}$$

The distance $d(r_i, r_j)$ is related to the overlap so that it varies from 0 (for identical range variables) to ∞ (for maximally distant range variables). Consequently, we define

$$R(r_i, r_j) = -\ln\left[\frac{\text{overlap}(r_i, r_j)}{\text{maxOverlap}}\right].$$

B From Rankings to Numerical Values

The numerical values in the tables $Z(\varkappa_i, \varkappa_j)$ can be formed from the buyer's rankings in a number of ways. Since we are minimizing distance, we want preferred options to contribute little Z and progressively less favorable options to have progressively larger Z values. To approach this, we first consider the problem of defining a Z' value normalized to lie between 0 and 1 and then take $Z = -\ln[1 - Z']$.

One possibility for Z' is a linear scaling. If d_i is the integer indicating the number of values in $\mathcal{D}_i$, $r_i^{(\alpha)}(\varkappa_j)$ indicates the buyer-specified rank (smaller rank means more preferable) for the option $\varkappa_i = \alpha \in \mathcal{D}_i$ conditional on $\varkappa_j$, then we might set $Z'(\varkappa_i = \alpha, \varkappa_j) = (r_i^{(\alpha)}(v_j) - 1)/(d_i - 1)$. Alternatively, we might use a non-linear scaling and set $Z(\varkappa_i = \alpha, v_j) = (1 - \beta^{r_i^{(\alpha)}(\varkappa_j)-1})/(1 - \beta^{d_i - 1})$ for some $\beta \in [0, 1]$ (exponential) or set $Z(v_i = \alpha, v_j) = ((r_i^{(\alpha)}(v_j) -$

type	Z'
linear	$\ln\left[\dfrac{R_i-1}{R_i-r_i^{(\alpha)}(v_j)}\right]$
exponential	$\ln\left[\dfrac{1-\beta^{R_i-1}}{\beta^{r_i^{(\alpha)}(v_j)-1}-\beta^{R_i-1}}\right]$
algebraic	$-\ln\left[1-\left(\dfrac{r_i^{(\alpha)}(v_j)-1}{R_i-1}\right)^n\right]$

Table 4: Possible numerical ranking functions.

$1)/(R_i-1))^n$ for some $n>0$ (algebraic). The corresponding scaling functions for Z are given in Table 4.

For those allowed values of $\varkappa_i$ which are left unranked by the buyer, we can assign them a common numerical value greater than the lowest ranked value. For example if $r_i(\varkappa_j)$ is the largest ranked value we might assign all remaining $\varkappa_i$ values to have $Z(\varkappa_i,\varkappa_j)=r_i(\varkappa_j)/(d_i-1)$ in the linear case or $Z(\varkappa_i,\varkappa_j)=(1-\beta^{r_i(\varkappa_j)})/(1-\beta^{d_i-1})$ or $Z(\varkappa_i,\varkappa_j)=(1-1/(r_i(\varkappa_j)+1))/(1-1/d_i)$ in the non-linear cases.

C Determination of Scaling Factors

In this section we supply the details necessary to determine the scaling factors Q_r and Q_d. In the equations defining the average distances we have need of the integrals $\int_V d\mathbf{u}\ \exp[-(\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})]$ and $\int_V d\mathbf{u}\ (\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})\exp[-(\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})]$. In general, these integrations are not analytically tractable; however we will assume that the region of integration defined by $\overline{x}$ and $\underline{x}$ extends at least 2 standard deviations out in all directions so that the volume of integration is effectively infinite. In this case we can evaluate the integrals in closed form:

$$\int_V d\mathbf{u}\ \exp[-(\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})] \approx \sqrt{\frac{\pi^{n_c}}{\det \mathbf{C}^{-1}}}$$

and

$$\int_V d\mathbf{u}\ (\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})\exp[-(\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})] \approx \frac{n_c}{2}\sqrt{\frac{\pi^{n_c}}{\det \mathbf{C}^{-1}}}.$$

To this point we have not been specific about what is meant by the average over the range variables. In fact, we will assume that, however we define this average, it is independent of what the buyer requested and thus can be removed from inside the sum over the discrete variables. In this case we will concentrate on setting the discrete and continuous components equal. We have $\langle d_c\rangle/n_c=\langle d_d\rangle/n_d$ where

$$\frac{\langle d_d\rangle}{n_d}\equiv\frac{Q_d}{n_d}\frac{\sum_{\mathbf{v}} d_d\exp[-Q_d d_d]\int_V d\mathbf{u}\ \exp[-d_c]}{\sum_{\mathbf{v}}\exp[-Q_d d_d]\int_V d\mathbf{u}\ \exp[-d_c]}\quad\text{and}\quad\frac{\langle d_c\rangle}{n_c}\equiv\frac{1}{n_c}\frac{\sum_{\mathbf{v}}\exp[-Q_d d_d]\int_V d\mathbf{u}\ d_c\exp[-d_c]}{\sum_{\mathbf{v}}\exp[-Q_d d_d]\int_V d\mathbf{u}\ \exp[-d_c]}$$

and cancelling denominators Q_d is determined by solving

$$\frac{Q_d}{n_d}\sum_{\mathbf{v}} d_d \exp[-Q_d d_d]\int_V d\mathbf{u}\ \exp[-d_c] = \frac{1}{n_c}\sum_{\mathbf{v}} \exp[-Q_d d_d]\int_V d\mathbf{u}\ d_c \exp[-d_c].$$

This can be simplified further given our assumptions about the Gaussian integral:

$$\frac{Q_d}{n_d}\sum_{\mathbf{v}} \frac{d_d \exp[-Q_d d_d]}{\sqrt{\det \mathbf{C}^{-1}(\mathbf{v})}} = \frac{1}{2}\sum_{\mathbf{v}} \frac{\exp[-Q_d d_d]}{\sqrt{\det \mathbf{C}^{-1}(\mathbf{v})}}.$$

In most cases there will also be at least one finite root, which is what we need. If there are a multiplicity of roots, we select that root having the smallest value to ensure that the distribution defined by the utility function is as flat as possible. If there are no real roots, then we select the Q_d which minimizes the error.

D Phase 1 Optimization

In this section we detail the solution to the phase 1 optimization problem defined in Eq. (5).

D.1 General Formulation

We begin by writing down the objective function:

$$d_1(\mathbf{x}, \boldsymbol{\varkappa}) = \left(\mathbf{x}^{(s)} - \boldsymbol{\mu}(\boldsymbol{\varkappa})\right)^t \mathbf{C}_w^{-1}\left(\mathbf{x}^{(s)} - \boldsymbol{\mu}(\boldsymbol{\varkappa})\right) + R(\mathbf{r}; \boldsymbol{\varkappa})$$

where (see Eq. (15))

$$\mathbf{x}^{(s)} = \left(\mathbf{I} - \mathbf{M}^{(s)}(\boldsymbol{\varkappa}^{(s)}, \{b_{i,k}^{(s)}\})\right)^{-1}\mathbf{c}(\boldsymbol{\varkappa}^{(s)}, \{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\}) + \left(\mathbf{I} - \mathbf{M}^{(s)}(\mathbf{v}^{(s)}, \{b_{i,k}^{(s)}\})\right)^{-1}\mathbf{M}^{(b)}(\{b_{i,k}^{(b)}\})\mathbf{u}^{(b)} \tag{15}$$

as long as $x_k^{(s)}(b_{i,k}^{(s)}) \le x_k^{(s)} \le x_k^{(s)}(b_{i,k}^{(s)} + 1)$. This must be optimized over $\mathbf{x}$ subject to the constraints $\mathbf{G}_1(\boldsymbol{\varkappa})\mathbf{x}^{(s)} = \mathbf{g}_1(\boldsymbol{\varkappa})$ and $\mathbf{G}_2(\boldsymbol{\varkappa})\mathbf{x}^{(s)} = \mathbf{g}_2(\boldsymbol{\varkappa})$ for a given $\boldsymbol{\varkappa}$. Since $\boldsymbol{\varkappa}$ is fixed throughout the optimization $R(\mathbf{r}; \boldsymbol{\varkappa})$ is just a constant which can be evaluated and then ignored as far as determination of $\mathbf{x}$ is concerned. We simplify our notation so that $d_1(\mathbf{u}) = \left(\mathbf{x}^{(s)} - \boldsymbol{\mu}\right)^t \mathbf{C}_w^{-1}\left(\mathbf{x}^{(s)} - \boldsymbol{\mu}\right) + R(\mathbf{r})$. Similarly,

$$\begin{aligned}\mathbf{x}^{(s)} &= \left(\mathbf{I} - \mathbf{M}^{(s)}(\{b_{i,k}^{(s)}\})\right)^{-1}\mathbf{c}(\{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\}) + \left(\mathbf{I} - \mathbf{M}^{(s)}(\{b_{i,k}^{(s)}\})\right)^{-1}\mathbf{M}^{(b)}(\{b_{i,k}^{(b)}\})\mathbf{x}^{(b)} \\ &\equiv \mathbf{s}(\{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\}) + \mathbf{S}(\{b_{i,k}^{(s)}, \{b_{i,k}^{(b)}\})\mathbf{x}^{(b)}\end{aligned}$$

where we have defined the n_c-vector $\mathbf{s} = (\mathbf{I} - \mathbf{M}^{(s)})^{-1}\mathbf{c}$ and the $n_c \times n_c'$ matrix $\mathbf{S} = (\mathbf{I} - \mathbf{M}^{(s)})^{-1}\mathbf{M}^{(b)}$.[28] $n_c' < n_c$ is the number of continuous dimensions upon which the supplier response depends. Putting these pieces together we have

$$d_1(\mathbf{x}^{(b)}) = \mathbf{x}^{(b)t}\mathbf{Q}(\{b_{i,k}^{(s)}, \{b_{i,k}^{(b)}\})\mathbf{x}^{(b)} + \mathbf{q}^t\mathbf{x}^{(b)} + const \tag{16}$$

[28] In the next section we consider $\mathbf{S}$ further, taking advantage of any redundant variables to simplify the QP task.

where $\mathbf{Q}$ is an $n'_c \times n'_c$ matrix given by

$$\mathbf{Q}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\}) = \mathbf{S}^t(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\})\mathbf{C}_w^{-1}\mathbf{S}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\})$$

$\mathbf{q}$ is an n'-vector given by

$$\mathbf{q}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\}) = 2\mathbf{S}^t(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\})\mathbf{C}_w^{-1}\big(\mathbf{s}\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\}) - \boldsymbol{\mu}\big)^t.$$

and the constant term is

$$const = \big(\mathbf{s}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\}) - -\boldsymbol{\mu}\big)^t\mathbf{C}_w^{-1}\big(\mathbf{s}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\}) - \boldsymbol{\mu}\big) + R(\mathbf{r})$$

It is easily shown that the matrix $\mathbf{Q}$ is positive semidefinite. $d_1(\mathbf{u}^{(b)})$ must be optimized subject to the constraints

$$\mathbf{G}_1\mathbf{S}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\})\mathbf{x}^{(b)} = \mathbf{g}_1 - \mathbf{G}_1\mathbf{s}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\}), \tag{17}$$

$$\mathbf{G}_2\mathbf{S}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\})\mathbf{x}^{(b)} \le \mathbf{g}_2 - \mathbf{G}_2\mathbf{s}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\}), \tag{18}$$

$$\underline{\mathbf{x}}^{(s)}(i) \le \mathbf{s}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\}) + \mathbf{S}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\})\mathbf{x}^{(b)} \le \overline{\mathbf{x}}^{(s)}(i) \;\; \forall\, i, \tag{19}$$

$$\underline{\mathbf{x}}^{(b)}(i) \le \mathbf{x}^{(b)} \le \overline{\mathbf{x}}^{(b)}(i) \;\; \forall\, i \tag{20}$$

where $\underline{\mathbf{x}}^{(s)}(i)^t = \Big[x_1(b^{(s)}_{i,1}) \;\; \cdots \;\; x_n(b^{(s)}_{i,n})\Big]$ and $\overline{\mathbf{x}}^{(s)}(i)^t = \Big[x_1(b^{(s)}_{i,1}+1) \;\; \cdots \;\; x_n(b^{(s)}_{i,n}+1)\Big]$ are the vectors of breakpoints for the $f^{(s)}_{i,k}$ $(k \in [1, n_c])$ functions. Similarly, for the buyer variables, $\underline{\mathbf{x}}^{(b)}(i)^t = \Big[x_1(b^{(b)}_{i,1}) \;\; \cdots \;\; x_n(b^{(b)}_{i,n})\Big]$ and $\overline{\mathbf{x}}^{(b)}(i)^t = \Big[x_1(b^{(b)}_{i,1}+1) \;\; \cdots \;\; x_n(b^{(b)}_{i,n}+1)\Big]$ are the vectors of breakpoints for the $f^{(b)}_{i,k}$ $(k \in [1, n_c])$ functions.

d_1 is optimized in two steps. First we assume a value for the integer variables $\{b^{(s)}_{i,k}\}$ and $\{b^{(b)}_{i,k}\}$ fixing the constraints in Eqs. (17) and (18). We then minimize the quadratic d_1 subject to these constraints using a standard quadratic programming (QP) algorithm[29] to find $\mathbf{x}^{(b)}_{\text{opt}}(\{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\})$. This result is then used to find $\mathbf{x}^*\big(\boldsymbol{\varkappa}; \{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\}\big) = \mathbf{s}\big(\{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\}\big) + \mathbf{S}\big(\{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\}\big)\mathbf{x}^{(b)}_{\text{opt}}\big(\{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\}\big)$. By searching exhaustively over $\{b^{(s)}_{i,k}\}$ and $\{b^{(b)}_{i,k}\}$ we determine

$$\mathbf{x}^{(s)}(\boldsymbol{\varkappa}) = \underset{\{b^{(s)}_{i,k}\},\{b^{(b)}_{i,k}\}}{\arg\min} \; d_1\big(\mathbf{x}^*\big(\boldsymbol{\varkappa}; \{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\}\big), \boldsymbol{\varkappa}\big)$$

Before turning to the manner in which this optimization is accomplished in section D.3, we discuss a few improvements to the input into the QP algorithm.

D.2 The Quadratic Program

In this section we discuss improvements in the specification of the quadratic programming task to make it better conditioned and more efficient.

It is possible that the matrix $\mathbf{Q}$ used as input to the QP is singular or near singular,

[29] Numerous third parties supply quadratic programming packages, some based on interior-point methods which are appropriate, in the present context where matrix $\mathbf{Q}$ may be singular.

depending on the rank of $\mathbf{S}_1$. Consequently, it is important to use an interior point QP algorithm for which this is not a problem.

D.2.1 Variable Elimination

There may be rows or columns of $\mathbf{S}$ which are entirely zero. It is important to take advantage of this structure to eliminate unnecessary computation.

If row i is zero, this indicates that supplier variable $u_i^{(s)}$ is independent of any buyer variable and is simply equal to s_i. If there are any constraints involving only supplier variable i, then these can be checked before calling the QP solver to eliminate wasteful computation.[30] If the constraint is satisfied, we can eliminate it and then call the QP function. If column j of $\mathbf{S}$ is all zero, this means that there is no dependence of $\mathbf{u}^{(s)}$ on $u_j^{(b)}$ and so there is no need to optimize over that buyer variable. We will, however, still have to check that the constraints on $\mathbf{u}^{(s)}$ are satisfied. This observation allows us to simplify the quadratic programming problem by eliminating this unnecessary variable.

We formulate the problem to take advantage of both these observations. Assume there are $m < n_c$ non-zero rows of $\mathbf{S}$ and $m' < n_c'$ non-zero columns. Relabel the rows and columns so that

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$

where $\mathbf{S}_1$ is of size $m \times m'$ and, with a slight abuse of notation, the $\mathbf{0}$'s indicate matrices of the appropriate size. In like fashion we write

$$\mathbf{x}^{(s)} = \begin{bmatrix} \mathbf{x}_1^{(s)} \\ \mathbf{x}_2^{(s)} \end{bmatrix}, \quad \mathbf{x}^{(b)} = \begin{bmatrix} \mathbf{x}_1^{(b)} \\ \mathbf{x}_2^{(b)} \end{bmatrix}$$

where $\mathbf{x}_1^{(s)}$ is an m vector, $\mathbf{x}_2^{(s)}$ is an $n_c - m$ vector, $\mathbf{x}_1^{(b)}$ is an m' vector, and $\mathbf{x}_2^{(b)}$ is an $n_c' - m'$ vector. $\mathbf{x}_1^{(s)}$ indicates those components of $\mathbf{x}^{(s)}$ which vary with buyer variables while $\mathbf{x}_2^{(s)}$ indicates those variables which are constant. $\mathbf{x}_1^{(b)}$ are those buyer variables which affect at least one supplier variable and $\mathbf{x}_2^{(b)}$ are those buyer variables which do not affect any supplier variables. If we also partition $\mathbf{s}$ as $\mathbf{s}^t = \begin{bmatrix} \mathbf{s}_1^t & \mathbf{s}_2^t \end{bmatrix}$ then

$$\mathbf{x}_1^{(s)} = \mathbf{s}_1 + \mathbf{S}_1 \mathbf{x}_1^{(b)} \quad \text{and} \quad \mathbf{x}_2^{(s)} = \mathbf{s}_2.$$

Finally, we will also have need of the partitioned covariance matrix

$$\mathbf{C}^{-1} = \begin{bmatrix} \mathbf{C}_{1,1}^{-1} & \mathbf{C}_{1,2}^{-1} \\ \mathbf{C}_{2,1}^{-1} & \mathbf{C}_{2,2}^{-1} \end{bmatrix}$$

where $(\mathbf{C}_{1,2;w}^{-1})^t = \mathbf{C}_{2,1;w}^{-1}$ since $\mathbf{C}_w^{-1}$ is symmetric.

[30] Additionally, the QP doesn't like to have a row of zeros in the constraint matrix.

Including the dependencies on μ, the quadratic form we must optimize is

$$\begin{bmatrix}(\mathbf{s}_1-\mu_1)^t+(\mathbf{x}_1^{(b)})^t\mathbf{S}_1^t & (\mathbf{s}_2-\mu_2)^t\end{bmatrix}\begin{bmatrix}\mathbf{W}_1\mathbf{C}_{1,1}^{-1}\mathbf{W}_1 & \mathbf{W}_1\mathbf{C}_{1,2}^{-1}\mathbf{W}_2\\ \mathbf{W}_2\mathbf{C}_{2,1}^{-1}\mathbf{W}_1 & \mathbf{W}_2\mathbf{C}_{2,2}^{-1}\mathbf{W}_2\end{bmatrix}\begin{bmatrix}\mathbf{s}_1-\mu_1+\mathbf{S}_1\mathbf{u}_1^{(b)}\\ \mathbf{s}_2-\mu_2\end{bmatrix}$$

where we have made the obvious definitions

$$\mathbf{W}=\begin{bmatrix}\mathbf{W}_1 & \mathbf{0}\\ \mathbf{0} & \mathbf{W}_2\end{bmatrix}\quad\text{and}\quad\mu=\begin{bmatrix}\mu_1\\ \mu_2\end{bmatrix}.$$

Expanding the quadratic, the constant term is

$$(\mathbf{s}_1-\mu_1)^t\mathbf{W}_1\mathbf{C}_{1,1}^{-1}\mathbf{W}_1(\mathbf{s}_1-\mu_1)+(\mathbf{s}_2-\mu_2)^t\mathbf{W}_2\mathbf{C}_{2,2}^{-1}\mathbf{W}_2(\mathbf{s}_2-\mu_2)+2(\mathbf{s}_1-\mu_1)^t\mathbf{W}_1\mathbf{C}_{1,2}^{-1}\mathbf{W}_2(\mathbf{s}_2-\mu_2).$$

The linear term is

$$\mathbf{q}=\left((\mathbf{s}_1-\mu_1)^t\mathbf{W}_1\mathbf{C}_{1,1}^{-1}\mathbf{W}_1\mathbf{S}_1^t+2(\mathbf{s}_2-\mu_2)^t\mathbf{W}_2\mathbf{C}_{2,1}^{-1}\mathbf{W}_1\mathbf{S}_1\right)\mathbf{x}_1^{(b)}$$

and the quadratic term is

$$\mathbf{Q}=(\mathbf{x}_1^{(b)})^t\mathbf{S}_1^t\mathbf{W}_1\mathbf{C}_{1,1}^{-1}\mathbf{W}_1\mathbf{S}_1\mathbf{x}_1^{(b)}.$$

The constraints involving the optimization variables $\mathbf{u}_1^{(b)}$ are

$$\begin{aligned}\mathbf{G}_{1;1}\mathbf{S}_1\mathbf{x}_1^{(b)}&=\mathbf{g}_1-\mathbf{G}_{1;1}\mathbf{s}_1-\mathbf{G}_{1;2}\mathbf{s}_2,\\ \mathbf{G}_{2;1}\mathbf{S}_1\mathbf{x}_1^{(b)}&\le\mathbf{g}_2-\mathbf{G}_{2;1}\mathbf{s}_1-\mathbf{G}_{2;2}\mathbf{s}_2,\\ \underline{\mathbf{x}}_1^{(s)}(i)\le\mathbf{s}_1+\mathbf{S}_1\mathbf{x}_1^{(b)}&\le\overline{\mathbf{x}}_1^{(s)}(i)\ \ \forall\, i,\\ \underline{\mathbf{x}}_1^{(b)}(i)\le\mathbf{x}_1^{(b)}&\le\overline{\mathbf{x}}_1^{(b)}(i)\ \ \forall\, i\end{aligned}$$

where we have simplified notation by dropping the dependence on breakpoints and defined

$$\mathbf{G}_i=\begin{bmatrix}\mathbf{G}_{i;1} & \mathbf{G}_{i;2}\end{bmatrix}$$

i.e., $\mathbf{G}_{i;1}$ is the columns of $\mathbf{G}_1$ involving the active buyer variables while $\mathbf{G}_{i;2}$ is formed from the inactive buyer variables. Similarly, the constraints which can be checked prior to even solving the QP are

$$\underline{\mathbf{x}}_2^{(s)}(i)\le\mathbf{s}_2\le\overline{\mathbf{x}}_2^{(s)}(i)\ \ \forall\, i.$$

D.2.2 QP Normalization

There are two types of normalizations we perform for increased numerical stability. If the scales of the continuous dimensions vary greatly (i.e., one dimension has an ideal value of 10^{-2} and another dimension has ideal value 10^6), then the matrix $\mathbf{C}^{-1}$ becomes badly conditioned, which makes $\mathbf{S}$ badly conditioned and thus makes the QP unstable. Similarly, if the breakpoints have

radically, different lengths, the QP problem can also become unstable. We consider how to remedy each of these problems.

Dimension normalization

The quadratic form to be minimized is $(\mathbf{x}-\boldsymbol{\mu})^t\mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu})$. We modify this problem by defining the dimensionless variables ϕ_i through $x_i = \phi_i\mu_i$ in terms of which

$$(\mathbf{x}-\boldsymbol{\mu})^t\mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu}) = (\boldsymbol{\phi}-\mathbf{1})^t\boldsymbol{\mathcal{C}}^{-1}(\boldsymbol{\phi}-\mathbf{1})$$

where $\boldsymbol{\mathcal{C}}^{-1} = \mathbf{D}\mathbf{C}^{-1}\mathbf{D}$ and $\mathbf{D} = \mathrm{diag}(\boldsymbol{\mu})$. When making this variable transformation we must assure that the constraints on $\mathbf{x}$ are adjusted so that $\mathbf{G}_1\mathbf{x} = \mathbf{g}_1$ and $\mathbf{G}_2\mathbf{x} \leq \mathbf{g}_2$ become the constraints $\boldsymbol{\mathcal{G}}_1\boldsymbol{\phi} = \mathbf{g}_1$ and $\boldsymbol{\mathcal{G}}_2\boldsymbol{\phi} \leq \mathbf{g}_2$ on $\boldsymbol{\phi}$ where $\boldsymbol{\mathcal{G}}_{1,2} = \mathbf{G}_{1,2}\mathbf{D}$. If there are global bounds of the form $\underline{\mathbf{x}} \leq \mathbf{x} \leq \overline{\mathbf{x}}$, then these are translated to $\underline{\boldsymbol{\phi}} \leq \boldsymbol{\phi} \leq \overline{\boldsymbol{\phi}}$ where $\underline{\boldsymbol{\phi}} = \mathbf{D}^{-1}\underline{\mathbf{u}}$ and $\overline{\boldsymbol{\phi}} = \mathbf{D}^{-1}\overline{\mathbf{u}}$.

In addition to these changes, we also need to modify the capabilities appropriately. Recall Eq. (2) which reads (after dropping the dependence on the discrete variables)

$$x_i^{(s)} = \sum_k f_{i,k}^{(s)}(x_k^{(s)}) + f_{i,k}^{(b)}(x_k^{(b)}). \tag{21}$$

In terms of the dimensionless variables we have

$$\phi_i^{(s)} = \sum_k \frac{f_{i,k}^{(s)}(\phi_k^{(s)}\mu_k)}{\mu_i} + \frac{f_{i,k}^{(b)}(\phi_k^{(b)}\mu_k)}{\mu_i}. \tag{22}$$

Thus, when we use the dimensionless variables, the function $f_{i,k}$ needs to be modified so that the y dimension is divided by μ_i and the x dimension is multiplied by μ_k.

Once the optimization has determined an optimal $\boldsymbol{\phi}$, we return to the original variables through $\mathbf{x} = \mathbf{D}\boldsymbol{\phi}$.

Breakpoint range normalization

For larger problems it is sometimes important to normalize the input variables for the QP solver. This is carried out in order to improve the QP search performance in situations where there is large variation in the range of the segments $\phi_k^{(b)}(b_k) \leq \phi_k^{(b)} \leq \phi_k^{(b)}(b_k+1)$ and the magnitude of their values. This can be accomplished by introducing the normalization variables $\hat{\phi}_k^{(b)} = (\phi_k^{(b)} - \phi_k^{(b)}(b_k))/(\phi_k^{(b)}(b_k+1) - \phi_k^{(b)}(b_k))$. Here $\hat{\phi}_k^{(b)}$ will have minimum value of zero and maximum of one. We can now express $\phi^{(b)}$ as

$$\boldsymbol{\phi}^{(b)} = \boldsymbol{\Lambda}\hat{\boldsymbol{\phi}}^{(b)} + \mathbf{p} \tag{23}$$

where $\boldsymbol{\Lambda}$ is the diagonal matrix $\boldsymbol{\Lambda} = \mathrm{diag}(\phi_1^{(b_1+1)} - \phi_1^{(b_1)}, \phi_2^{(b_2+1)} - \phi_2^{(b_2)}, \cdots, \phi_m^{(b_m+1)} - \phi_m^{(b_m)})$ and $\mathbf{p}^t = [\phi_1^{(b_1)}, \phi_2^{(b_2)}, \cdots, \phi_m^{(b_m)}]$.

Expanding Eq.(16) using the definition of $\boldsymbol{\phi}$ in Eq.(23), we obtain

$$d_1(\mathbf{w};\{b_{i,k}\}) = \frac{1}{2}\hat{\boldsymbol{\phi}}^{(b)t}\boldsymbol{\Lambda}\mathbf{D}\mathbf{Q}(\{b_{i,k}\})\mathbf{D}\boldsymbol{\Lambda}\hat{\boldsymbol{\phi}}^{(b)} + \mathbf{q}^t(\{b_{i,k}\})\mathbf{D}\boldsymbol{\Lambda}\hat{\boldsymbol{\phi}}^{b} + \mathbf{p}^t\mathbf{D}\mathbf{Q}(\{b_{i,k}\})\mathbf{D}\boldsymbol{\Lambda}\hat{\boldsymbol{\phi}}^{(b)} + \frac{1}{2}\mathbf{p}^t\mathbf{D}\mathbf{Q}(\{b_{i,k}\})\mathbf{D}\mathbf{p} + \mathbf{q}^t(\{b_{i,k}\})\mathbf{D}\mathbf{p}.$$

Hence the quadratic programming problem to be solved in terms of $\hat{\boldsymbol{\phi}}$ is

$$QP(\hat{\mathbf{x}}^{(b)};\{b_{i,k}\}) = \frac{1}{2}\hat{\mathbf{x}}^{(b)t}\hat{\mathbf{Q}}(\{b_{i,k}\})\hat{\mathbf{x}}^{(b)} + \hat{\mathbf{q}}^t(\{b_{i,k}\})\hat{\mathbf{u}}^{(b)}, \tag{24}$$

where $\hat{\mathbf{Q}} = \boldsymbol{\Lambda}\mathbf{D}\mathbf{Q}(\{b_{i,k}\})\mathbf{D}\boldsymbol{\Lambda}$ and $\hat{\mathbf{q}}^t = \big(\mathbf{q}^t(\{b_{i,k}\}) + \mathbf{p}^t\mathbf{D}\mathbf{Q}(\{b_{i,k}\})\big)\mathbf{D}\boldsymbol{\Lambda}$. Similarly, the equality and inequality constraints of Eqs. (17) and (18) expressed in terms of $\hat{\boldsymbol{\phi}}$ are

$$\hat{\mathbf{G}}_1(\{b_{i,k}\})\hat{\boldsymbol{\phi}}^{(b)} = \hat{\mathbf{g}}_1 \quad \text{and} \quad \hat{\mathbf{G}}_2(\{b_{i,k}\})\hat{\boldsymbol{\phi}}^{(b)} \leq \hat{\mathbf{g}}_2, \tag{25}$$

where

$$\begin{aligned}
\hat{\mathbf{G}}_1(\{b_{i,k}\}) &= \mathbf{G}_1\mathbf{S}(\{b_{i,k}\})\mathbf{D}\boldsymbol{\Lambda}, \\
\hat{\mathbf{G}}_2(\{b_{i,k}\}) &= \mathbf{G}_2\mathbf{S}(\{b_{i,k}\})\mathbf{D}\boldsymbol{\Lambda}, \\
\hat{\mathbf{g}}_1 &= \mathbf{g}_1 - \mathbf{G}_1\mathbf{s}(\{b_{i,k}\}) - \mathbf{G}_1\mathbf{S}(\{b_{i,k}\})\mathbf{D}\mathbf{p}, \\
\hat{\mathbf{g}}_2 &= \mathbf{g}_2 - \mathbf{G}_2\mathbf{s}(\{b_{i,k}\}) - \mathbf{G}_1\mathbf{S}(\{b_{i,k}\})\mathbf{D}\mathbf{p}.
\end{aligned}$$

D.3 Optimization Over The Breakpoint Variables

We have detailed how to determine $\mathbf{x}^{(b)}_{\text{opt}}(\boldsymbol{\varkappa}, \{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\})$ given a specification of all $\{b^{(s)}_{i,k}\}$ and $\{b^{(b)}_{i,k}\}$, but how do we determine the best feasible choice of $\{b^{(s)}_{i,k}\}$ and $\{b^{(b)}_{i,k}\}$? Before we answer that question, we have to examine the infeasibilities that might arise due to improper choices of $\{b^{(s)}_{i,k}\}$ and $\{b^{(b)}_{i,k}\}$.

D.3.1 Elimination of $\{b_{i,k}\}$ Infeasibilities

Most of the settings of $\{b^{(b)}_{i,k}\}$ or $\{b^{(s)}_{i,k}\}$ will not yield a feasible solution because Eq. (20) or Eq. (19) will be violated. The reason is that, for any k, it is highly likely to get two segments $x^{(b)}_k(b_{i,k}) \leq x^{(b)}_k \leq x^{(b)}_k(b_{i,k}+1)$ and $x^{(b)}_k(b_{j,k}) \leq x^{(b)}_k \leq x^{(b)}_k(b_{j,k}+1)$ that do not overlap (i.e., the optimizer will not be able to find a $x^{(b)}_k$ value that satisfies both bounds, and hence no feasible solution will be found). The same is also true for supplier response segment $x^{(s)}_k$.

To eliminate those infeasibilities, we construct a super set of breakpoints $\{\varpi^b_k\}$ such that

$$\varpi^{(b)}_k = \bigcup_{i=1}^{n}\{x^{(b)}_k(b^{(b)}_{i,k}=1), x^{(b)}_k(b^{(s)}_{i,k}=2), \cdots, x^{(b)}_k(b^{(b)}_{i,k}=\kappa^{(b)}_{i,k})\}$$

Hence each piecewise linear function $f_{i,k}$ will be described by a set of breakpoints $\varpi_k = \{x_k(1), \cdots, x_k(\kappa_k)\}$ and function values $f_{i,k}(x_k(1)), \cdots, f_{i,k}(x_k(\kappa_k))$. These function values

at the new breakpoints are linearly interpolated from the old breakpoint-function pairs. After this reduction we have $\kappa_k - 1$ different $b_k \in [1, \kappa_k - 1]$ values, which eliminates the need for constraint Eq. (20).

The continuous supplier variables and constraint Eq. (19) are more problematic. Suppressing any dependence on $\boldsymbol{\varkappa}$, we know

$$x_i^{(s)} = c_i(\{b_{i,1}^{(s)}, \cdots, b_{i,i-1}^{(s)}\}, \{b_{i,k}^{(b)}\}) + \sum_{k<i} m_{i,k}^{(s)}(b_{i,k}^{(s)}) x_k^{(s)} + \sum_k m_{i,k}^{(b)}(b_{i,k}^{(b)}) x_k^{(b)}.$$

If $\{b_{i,k}^{(b)}\}$ and $\{b_{i,1}^{(s)}, \cdots, b_{i,i-1}^{(s)}\}$ are known, then this expression allows us to bound the values $x_i^{(s)}$ may take on, and thus bound the interval in which the feasible $b_{i,i}^{(s)}$ lie. If we define the functions

$$\mathrm{smax}(s,x,y) = \begin{cases} s\max(x,y) & \text{if } s \geq 0 \\ s\min(x,y) & \text{if } s < 0 \end{cases} \quad \text{and} \quad \mathrm{smin}(s,x,y) = \begin{cases} s\min(x,y) & \text{if } s \geq 0 \\ s\max(x,y) & \text{if } s < 0 \end{cases}$$

then the minimum and maximum values that $x_i^{(s)}$ may take on are:

$$\begin{aligned} \min(x_i^{(s)}) = c_i(\{b_{i,1}^{(s)}, \cdots, b_{i,i-1}^{(s)}\}, \{b_{i,k}^{(b)}\}) &+ \sum_{k<i} \mathrm{smin}\big(m_{i,k}^{(s)}(b_{i,k}^{(s)}), x_k^{(s)}(b_{i,k}^{(s)}), x_k^{(s)}(b_{i,k}^{(s)}+1)\big) \\ &+ \sum_k \mathrm{smin}\big(m_{i,k}^{(b)}(b_{i,k}^{(b)}), x_k^{(b)}(b_{i,k}^{(b)}), x_k^{(b)}(b_{i,k}^{(b)}+1)\big), \end{aligned} \tag{26}$$

and

$$\begin{aligned} \max(x_i^{(s)}) = c_i(\{b_{i,1}^{(s)}, \cdots, b_{i,i-1}^{(s)}\}, \{b_{i,k}^{(b)}\}) &+ \sum_{k<i} \mathrm{smax}\big(m_{i,k}^{(s)}(b_{i,k}^{(s)}), x_k^{(s)}(b_{i,k}^{(s)}), x_k^{(s)}(b_{i,k}^{(s)}+1)\big) \\ &+ \sum_k \mathrm{smax}\big(m_{i,k}^{(b)}(b_{i,k}^{(b)}), x_k^{(b)}(b_{i,k}^{(b)}), x_k^{(b)}(b_{i,k}^{(b)}+1)\big) \end{aligned} \tag{27}$$

respectively. The above equations imply that we can fully translate a complete specification of buyer variable breakpoint intervals to a corresponding set of feasible supplier variable breakpoint intervals. We will search over the breakpoints using a branch-and-bound algorithm, and will need to be able to infer the restriction on the values of feasible supplier breakpoint intervals given a partial specification of buyer breakpoint interval variables. This inference, which narrows the domains of the supplier breakpoint variables, is called constraint propagation. Before describing how the branch-and-bound algorithm search over breakpoint interval variables is accomplished, we show how the constraint propagation may be obtained.

Narrowing of Supplier Breakpoint Domains Through Propagation

Let $\boldsymbol{\beta}_k^{(b)}$ be a vector of integers specifying a set of breakpoint intervals for the kth buyer variable. $|\boldsymbol{\beta}_k| = 1$ if the interval is uniquely determined, and $|\boldsymbol{\beta}_k| = \kappa_k^{(b)} - 1$ if no restrictions on the interval have been specified. We index the lth component of $\boldsymbol{\beta}_k^{(b)}$ as $\beta_{k;l}^{(b)}$, $l \in [1, \kappa_k^{(b)} - 1]$. A set of breakpoints for all buyer dimensions is written as $\{\boldsymbol{\beta}_k^{(b)}\}$. In analogous fashion, we define $\boldsymbol{\beta}_k^{(s)}$, $\beta_{k;l}^{(s)}$, $l \in [1, \kappa_k^{(s)} - 1]$, and $\{\boldsymbol{\beta}_k\}$ for the supplier breakpoint interval variables.

From a specification $\{\boldsymbol{\beta}_k^{(b)}\}$ we infer the feasible domains $\{\boldsymbol{\beta}_k^{(s)}\}$ using $\max(x_i^{(s)})$ and $\min(x_i^{(s)})$ where:

$$\max(x_i^{(s)}) = \sum_k \max_{b_k^{(b)} \in \boldsymbol{\beta}_k^{(b)}} \left\{ c_{i,k}(\{b_k^{(b)}\}) + \operatorname{smax}\left(m_{i,k}^{(b)}(b_k^{(b)}), x_k^{(b)}(b_k^{(b)}), x_k^{(b)}(b_k^{(b)}+1)\right) \right\} + \sum_{k<i} \max_{b_k^{(s)} \in \boldsymbol{\beta}_k^{(s)}} \left\{ c_{i,k}(\{b_k^{(s)}\}) + \operatorname{smax}\left(m_{i,k}^{(s)}(b_k^{(s)}), x_k^{(s)}(b_k^{(s)}), x_k^{(s)}(b_k^{(s)}+1)\right) \right\} \quad (28)$$

and

$$\min(x_i^{(s)}) = \sum_k \min_{b_k^{(b)} \in \boldsymbol{\beta}_k^{(b)}} \left\{ c_{i,k}(\{b_k^{(b)}\}) + \operatorname{smin}\left(m_{i,k}^{(b)}(b_k^{(b)}), x_k^{(b)}(b_k^{(b)}), x_k^{(b)}(b_k^{(b)}+1)\right) \right\} + \sum_{k<i} \min_{b_k^{(s)} \in \boldsymbol{\beta}_k^{(s)}} \left\{ c_{i,k}(\{b_k^{(s)}\}) + \operatorname{smin}\left(m_{i,k}^{(s)}(b_k^{(s)}), x_k^{(s)}(b_k^{(s)}), x_k^{(s)}(b_k^{(s)}+1)\right) \right\} \quad (29)$$

We begin with $i = 1$ (where no supplier variables contribute to $x_1^{(s)}$) and determine the interval $I_1 = [\min(x_1^{(s)}), \max(x_1^{(s)})]$. We then form the vector $\boldsymbol{\beta}_1^{(s)}$ with components $\{\beta_{1;l}^{(s)} | [x_1^{(s)}(\beta_{1;l}), x_1^{(s)}(\beta_{1;l}+1)] \cap I_1 \neq \emptyset\}$. Equations (28) and (29) can then be iterated to find the interval $I_2 = [\min(x_2^{(s)}), \max(x_2^{(s)})]$ using the previously calculated $\min(x_1^{(s)})$ and $\max(x_1^{(s)})$. With I_2 known, the components of $\boldsymbol{\beta}_2^{(s)}$ are determined as above. Iterating this procedure, we can find the potentially narrowed domains of all $\{\beta_k^{(s)}\}$.

We now return to the optimization problem: we need to search over all $\prod_k \kappa_k^{(b)} \times \prod_k \kappa_k^{(s)}$ combinations of breakpoints to determine the combination that minimizes $d_1\left(\mathbf{x}_{\text{opt}}(\boldsymbol{\varkappa}, \{b_k^{(b)}\}, \{b_k^{(s)}\}), \boldsymbol{\varkappa}\right)$. The simplest manner in which this accomplished is via exhaustive enumeration over all $\{b_k^{(b)}\}$ and $\{b_k^{(s)}\}$. However, once all $b_k^{(b)}$ have been specified, we need only search exhaustively over the corresponding $\{\beta_k(s)\}$ inferred as above. There are, however, many ways to speed up the search over the buyer breakpoint intervals. We describe two methods.

D.3.2 Local Search

This method is extremely fast but not guaranteed to return the global optimum across all buyer breakpoint intervals. Here, initial random and feasible[31] $\{b_k^{(s)}\}$, and $\{b_k^{(b)}\}$ sets are first created. After the continuous optimization is carried out using the initial $\{b_k^{(s)}\}$, and $\{b_k^{(b)}\}$, a new $\{\tilde{b}_k^{(s)}\}$, and $\{\tilde{b}_k^{(b)}\}$ set is generated from the old one by examining whether each generated $x_k^{(b)}(\{b_k^{(s)}\}, \{b_k^{(b)}\})$ solution is on the boundaries of the interval $x_k^{(b)}(b_k) \leq x_k^{(b)} \leq x_k^{(b)}(b_k+1)$ or $x_k^{(s)}(b_k) \leq x_k^{(s)} \leq x_k^{(s)}(b_k+1)$. If $x_k^{(b)}$ is on the lower bound of the interval, $\tilde{b}_k^{(b)} = b_k^{(b)} - 1$; similarly, if it is on the upper bound, $\tilde{b}_k^{(b)} = b_k^{(b)} + 1$. If $x_k^{(b)}$ is not on either end, then $\tilde{b}_k^{(b)} = b_k^{(b)}$. Of course this done while maintaining $\tilde{b}_k^{(b)}$ in the interval $1 \leq \tilde{b}_k^{(b)} \leq \kappa_k^{(b)} - 1$. Supplier breakpoint variables are updated in a similar manner. The procedure is repeated until no decrease in distance is attained.

[31] Feasible in the sense that they do not violate the constraints in equations (17), (18), (19), and (20).

D.3.3 Branch-and-Bound Search

Branch-and-bound is an efficient procedure used to exhaustively search a space and eliminate configurations that can be proven to be suboptimal. The theory behind branch-and-bound is well understood [5]. The idea is to structure the space as a tree of partially-completed possibilities and to explore the tree so that good complete solutions are constructed early on. Branches of the tree that can be shown to yield solutions which cannot be better than the currently best solution need not be explored. Heuristics indicate the order in which to explore the tree and provide a lower bound on the fitness of partially completed solutions for pruning of suboptimal branches..

In the present context, we carry out branch-and-bound search over the space of possible buyer breakpoint combinations. The tree is structured beginning from a node at which none of the $b_k^{(b)}$ have been assigned. From this node, the tree branches into the $\kappa_i^{(b)} - 1$ possible assignations for $b_i^{(b)}$. From any of these nodes, the tree branches into the $\kappa_j^{(b)}$ possible values for $b_j^{(b)}$. The tree branches at each node to assign the remaining variables. At the leaves of the tree, all buyer breakpoint variables have been assigned.

From any node, the search proceeds by choosing the most promising node to consider, as indicated by a variable ordering heuristic and considering the possible values for that variable in the order indicated by the value ordering heuristic. A complete solution and its accompanying distance is obtained when the search ends at a leaf of the tree. To avoid unnecessary exploration, a lower bound heuristic indicates when a partially completed solution (node) need not be expanded because all completions will have higher distance than the best located so far. We describe these three heuristics for search over buyer breakpoints.

Lower Bound

Given a set of buyer breakpoint interval vectors $\{\boldsymbol{\beta}_k^{(b)}\}$, we can propagate to $\{\boldsymbol{\beta}^{(s)}\}$ and the intervals $[\min(x_i^{(s)}), \max(x_i^{(s)})]$. A lower bound on the value d_1 can be obtained by ignoring the supplier capabilities and minimizing

$$d_1(\mathbf{x}) = \left(\mathbf{x}^{(s)} - \boldsymbol{\mu}\right)^t \mathbf{C}_w^{-1} \left(\mathbf{x}^{(s)} - \boldsymbol{\mu}\right) \tag{30}$$

subject to

$$\mathbf{G}_1\mathbf{x}^{(\mathbf{s})} = \mathbf{g}_1 \tag{31}$$

$$\mathbf{G}_2\mathbf{x}^{(\mathbf{s})} \leq \mathbf{g}_2 \tag{32}$$

$$\min(x_i^{(s)}) \leq x_i^{(s)} \leq \max(x_i^{(s)}), \tag{33}$$

The lower bound thus obtained is denoted by $\mathrm{lb}(\{\boldsymbol{\beta}^{(b)}\})$.

Variable and Value Ordering

The lower bound $\mathrm{lb}(\{\boldsymbol{\beta}^{(b)}\})$ can be used to define an ordering over variables and the values any variable may take on. We define the functions $L_{k,l}$ which operate on the set $\{\boldsymbol{\beta}^{(b)}\}$ by locking the kth buyer breakpoint variable to $\beta_{k;l}$. At any node we can determine the lower

bounds $\mathrm{lb}\big(L_{k,l}\{\boldsymbol{\beta}^{(b)}\}\big)$. We can select to explore the variables in order of increasing lb_k where

$$\mathrm{lb}_k(\{\boldsymbol{\beta}^{(b)}\}) = \frac{\sum_{l=1}^{|\boldsymbol{\beta}_k^{(b)}|} \mathrm{lb}\big(L_{k,l}(\{\boldsymbol{\beta}^{(b)}\})\big)}{\sum_{l=1}^{|\boldsymbol{\beta}_k^{(b)}|} 1},$$

i.e., the average over all possible completions for $b_k^{(b)}$.[32] Within any given variable the (say k^*) values can be explored in order of increasing $\mathrm{lb}\big(L_{k^*,l}(\{\boldsymbol{\beta}^{(b)}\})\big)$. If we are willing to do more computation, we can explore the branches in an interleaved order according to all the $\mathrm{lb}\big(L_{k,l}(\{\boldsymbol{\beta}^{(b)}\})\big)$.

Pruning and Search

During the branch-and-bound search, branches can be pruned if the node has $\mathrm{lb}\big(L_{k,l}(\{\boldsymbol{\beta}^{(b)}\})\big)$ greater than the distance of the lowest examined leaf node.

Search begins at the root node where each $\boldsymbol{\beta}_k^{(b)} = [1, \cdots, \kappa_k^{(b)} - 1]$. We applying the ordering heuristics above to move to a node where one breakpoint variable has been fixed. From that node, we move down the tree, fixing a value at a time until we reach a leaf. We then progressively backtrack up the tree, eliminating branches whose lower bounds are greater than the best leaf examined.

Once the optimal set of buyer breakpoints has been determined, we search exhaustively over the supplier breakpoints feasible with this choice according to Equations (26) and (27).

Improvements

We have not yet exploited the fact that, from any node having $\{\boldsymbol{\beta}_k^{(b)}\}$, the constraints in Equation (19) may narrow the domains of the buyer breakpoint variables deeper in the tree. The computational cost of narrowing the breakpoint domains may be warranted if the pruning thus obtained is sufficient.

E Phase 2 Optimization

In this section we detail an efficient algorithm to optimize Eq. (5):

$$\arg\min_{\boldsymbol{\varkappa}} \; d_1(\mathbf{x}(\boldsymbol{\varkappa}), \boldsymbol{\varkappa}) + d_2(\boldsymbol{\varkappa}) \quad \text{subject to} \quad \mathcal{C}^{(b)}(\boldsymbol{\varkappa}) \wedge \mathcal{C}^{(s)}(\boldsymbol{\varkappa}).$$

Our approach is a marriage of constraint programming and local search.

E.1 Local Search over *D*-neighborhoods

We begin with a definition. The d-neighbors of a discrete configuration $\boldsymbol{\varkappa}$ are all those configurations, $\boldsymbol{\varkappa}'$, which differ from $\boldsymbol{\varkappa}$ in d attributes. Attribute i differs in two configurations if $\varkappa_i \neq \varkappa_i'$. Thus the distance varies from 0 if the configurations are identical, up to n_d if all attributes are different. The D-neighborhood of $\boldsymbol{\varkappa}$ is the union of all $d = 1$ neighbors, $d = 2$

[32] We might also take the minimum over all possible completions $\mathrm{lb}_k(\{\boldsymbol{\beta}^{(b)}\}) = \min_{l \in [1, |\boldsymbol{\beta}_k^{(b)}|]} \mathrm{lb}\big(L_{k,l}(\{\boldsymbol{\beta}^{(b)}\})\big)$.

1	***11
2	**1*1
3	**11*
4	*1**1
5	*1*1*
6	*11**
7	1***1
8	1**1*
9	1*1**
10	11***

(a)

1	***+1
2	**1*1
3	**1+*
4	*+**1
5	*+*+*
6	*+1**
7	1***1
8	1**+*
9	1*1**
10	1+***

(b)

Table 5: Configurations examined in the $D = 3$ neighborhood search around (a) the initial configuration 11111 and (b) the initial configuration 0+0+0 (duplicate search configurations are indicated in bold) .

neighbors, up to $d = D$ neighbors. The D-neighborhood around $\boldsymbol{\varkappa}$ is indicated by $\mathcal{N}_D(\boldsymbol{\varkappa})$. We note that, for any configuration, its n_d neighborhood is the entire discrete space.

Using the D-neighborhood, we apply a standard hillclimbing algorithm which respects all the discrete constraints. D is specified before the algorithm begins and serves as an efficient manner to trade off running time with the quality of the final solution. The hillclimbing algorithm takes as input a starting configuration $\boldsymbol{\varkappa}_t$ and finds the configuration $\boldsymbol{\varkappa}_{t+1} \in \mathcal{N}_D(\boldsymbol{\varkappa}_t)$ which minimizes $d_1(\mathbf{x}(\boldsymbol{\varkappa}), \boldsymbol{\varkappa}) + d_2(\boldsymbol{\varkappa})$ subject to $\mathcal{C}^{(b)}(\boldsymbol{\varkappa}) \wedge \mathcal{C}^{(s)}(\boldsymbol{\varkappa})$. The algorithm then iterates again starting from $\boldsymbol{\varkappa}_{t+1}$. The hillclimbing algorithm terminates when it finds a configuration $\boldsymbol{\varkappa}_T$ which has a distance less than or equal to all of its D-neighbors. If $D = n_d$, this is the global minimum. Given a function which exhaustively searches a D-neighborhood for the lowest distance feasible configuration, a D-neighborhood hillclimbing algorithm is easily implemented.

There is one noteworthy point about the implementation of an efficient D-neighborhood hillclimber. Consider a $D = 3$ search over $n_d = 5$ discrete variables. Without loss of generality, we label the initial configuration as 11111. In the $D = 3$ neighborhood search, the configurations examined are given in Table 5(a) as indicated by the *s. Let us assume that the best neighbor in the D neighborhood of 11111 differs in components 2 and 4. We indicate this best neighbor as 1+1+1, which serves as the starting configuration for a new $D = 3$ search. The configurations examined in the second neighborhood search are given in Table 5(b). From Table 5(b) we see that the search branches 2, 7, and 10 are redundant, having already been performed in the D neighborhood search around 11111.[33] Thus these branches may be pruned in the second D neighborhood search. A search branch covers a previous search and may be pruned if the vector of positions of the wildcard characters covers the vector of positions at

[33]The duplication of search effort is most acute when D is large and the fittest neighbor is found very close by (small d).

which the previous best solution was found. The binary relation covers is defined as

$$\boldsymbol{\varkappa} \text{ covers } \boldsymbol{\varkappa}' \iff \varkappa'_i \in \boldsymbol{\varkappa} \ \forall \ \varkappa'_i \in \boldsymbol{\varkappa}'.$$

The covers relation is transitive but not reflexive. Thus, to increase the efficiency, we can keep a list of the previous branches searched and ignored search branches that have already been covered.

E.1.1 Improvements

Throughout the present discussion we have assumed that D remains constant through the hillclimbing process. There are good reasons why this should not be the case. With any reasonable landscape correlation structure, we know that lower distance neighbors are more likely to be found early in the hillclimbing process. Thus a reasonable heuristic is to start with a larger D early on and gradually lower it with time.

E.2 Exhaustive Search of a D-neighborhood

To complete the discussion of the phase 2 optimization, we describe a computationally efficient way in which to find the lowest distance feasible configuration within a D-neighborhood. We use a branch-and-bound method with heuristics tuned for the present problem. The factors tuned include the variable order in which we search, the order of the values any given variable may take, and a heuristic which says when certain branches of the search space may be safely ignored.

We use heuristics which early on in the search generate low-distance configurations, and later prune parts of the search that we can prove must have higher distance than the best configuration generated so far.

E.2.1 Heuristics And Lower Bounds

We focus on obtaining good heuristics for the discrete contribution to the total distance, then later incorporate the continuous and range contributions since these are simpler.

At any point in the search, some variables have been assigned and some have not. We divide up the sum over the $Z_{i,j}(\varkappa_i, \varkappa_j)$ into four contributions

$$\begin{aligned} Q_d Z_w(\boldsymbol{\varkappa}) = {} & Q_d \sum_{i \text{ locked}} \sum_{j \text{ locked}} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j) + Q_d \sum_{i \text{ locked}} \sum_{j \text{ floating}} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j) \\ & + Q_d \sum_{i \text{ floating}} \sum_{j \text{ locked}} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j) + Q_d \sum_{i \text{ floating}} \sum_{j \text{ floating}} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j) \\ \equiv {} & Z_{\mathrm{l,l}} + Z_{\mathrm{l,f}} + Z_{\mathrm{f,l}} + Z_{\mathrm{f,f}} \end{aligned}$$

Since we are minimizing and all $Z_{i,j}(\varkappa_i, \varkappa_j)$, w_i, and Q_d are positive, we can prune any search node which has $Z_{\mathrm{l,l}} >$ than the distance of the lowest distance configuration found thus far, since additional terms will only add to the sum.

We obtain a lower bound on the distance by lower-bounding each of the above four terms. Just taking the lowest value in the tables can yield differing values for any given variable, e.g., the values for a single variable $\varkappa_i$:

$$\varkappa_i^* = \arg\min_{\varkappa_i} Z_{i,j}(\varkappa_i, \varkappa_j), \quad \varkappa_i^* = \arg\min_{\varkappa_i} Z_{j,i}(\varkappa_j, \varkappa_i), \quad \text{and} \quad \varkappa_i^* = \arg\min_{\varkappa_i} Z_{i,i}(\varkappa_i)$$

may all be different. To correct this we write

$$Q_d Z_w = Z_{\mathrm{l,l}} + Z_{\mathrm{f,f}} + \sum_{i \text{ floating}} Z_{\mathrm{l,f}}^{(i)}(\varkappa_i^*)$$

where $Z_{\mathrm{l,l}}$ and $Z_{\mathrm{f,f}}$ are calculated as before and

$$Z_{\mathrm{l,f}}^{(i)}(\varkappa_i) = Q_d\Big(w_i^2 Z_{i,i}(\varkappa_i) + w_i \sum_{j \text{ locked}} w_j\big(Z_{i,j}(\varkappa_i, \varkappa_j) + Z_{j,i}(\varkappa_j, \varkappa_i)\big)\Big).$$

The best choice for the floating variable $\varkappa_i$ is

$$\varkappa_i^* = \arg\min_{\varkappa_i} Z_{\mathrm{l,f}}^{(i)}(\varkappa_i). \tag{34}$$

In this way all the possible choices for $\varkappa_i$ are at least consistent, and the true lower limit is still guaranteed to be higher than this (since the $Z_{\mathrm{f,f}}$ contribution can only be higher).

This approach also naturally suggests a variable ordering and a value ordering within any variable. We can try the variables in increasing order of their corresponding $Z_{\mathrm{l,f}}^{(i)}(\varkappa_i^*)$ so that variable i, which has the smallest $Z_{\mathrm{l,f}}^{(i)}(\varkappa_i^*)$, is tried first. For any given variable $\varkappa_i$ we can also order the possible value assignations for $\varkappa_i$ according to the array of values $Z_{\mathrm{l,f}}^{(i)}(\varkappa_i)$ so that the value $\varkappa_i = \varkappa_i^*$ is tried first and the value which maximizes $Z_{\mathrm{l,f}}^{(i)}(\varkappa_i)$ is tried last.

With little extra work we can impose a consistency requirement raising the lower bound for $Z_{\mathrm{f,f}}$. We decompose the sum as follows

$$Z_{\mathrm{f,f}} \equiv Q_d \sum_{i,j \text{ floating}: j \neq i} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j) = Q_d \sum_{i,j \text{ floating}: j < i} w_i w_j\big(Z_{i,j}(\varkappa_i, \varkappa_j) + Z_{j,i}(\varkappa_j, \varkappa_i)\big).$$

Instead of taking the minimum component-wise as previously suggested, *i.e* instead of

$$Z_{\mathrm{f,f}}^* = Q_d \sum_{i,j \text{ floating}: j \neq i} \min_{\varkappa_i, \varkappa_j}\big[w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j)\big],$$

we take the minimum so that

$$Z_{\mathrm{f,f}}^* = Q_d \sum_{i,j \text{ floating}: j < i} \min\big[w_i w_j\big(Z_{i,j}(\varkappa_i, \varkappa_j) + Z_{j,i}(\varkappa_j, \varkappa_i)\big)\big]. \tag{35}$$

which enforces pairwise consistency between variables i and j.

The lower bound, lb, we have obtained is thus

$$\text{lb} = Z_{\text{l,l}} + Z^*_{\text{f,f}} + \sum_i Z^{(i)}_{\text{l,f}}(\varkappa^*_i)$$

where $\varkappa^*_i$ is determined from Eq. (34) and $Z^*_{\text{f,f}}$ is determined from Eq. (35).

Integration Including Continuous and Range Contributions

In this section we summarize the complete heuristic specification, including the effects of the continuous and range variables. As above, we use the definitions

$$Z_{\text{l,l}} = Q_d \sum_{i,j \text{ locked}} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j)$$

$$Z^{(i)}_{\text{l,f}}(\varkappa_i) = Q_d w_i \sum_{j \text{ locked}} w_j \big(Z_{i,j}(\varkappa_i, \varkappa_j) + Z_{j,i}(\varkappa_j, \varkappa_i)\big)$$

$$Z^*_{\text{f,f}} = Q_d \sum_{i,j \text{ floating}:j<i} \min\big[w_i w_j \big(Z_{i,j}(\varkappa_i, \varkappa_j) + Z_{j,i}(\varkappa_i, \varkappa_j)\big)\big]$$

so that a lower bound to the complete discrete contribution to the distance is $Z_{\text{l,l}} + Z^*_{\text{f,f}} + \sum_i Z^*_{\text{l,f}}(\varkappa^*_i)$ where $\varkappa^*_i = \arg\min_{\varkappa_i} Z^{(i)}_{\text{l,f}}(\varkappa_i)$.

We now extend this result to include the continuous and range variables. The continuous and range variables depend on the discrete variables through the mask. Since we expect there will be relatively few masks we exhaustively enumerate all the discrete possibilities. Consequently, we define $d_1(\boldsymbol{\varkappa}_{\text{floating}}|\boldsymbol{\varkappa}_{\text{locked}})$ as the distance contribution made by the continuous and range variables (the d_1 contribution to Eq. (5)). We define the continuous + range contribution for any given floating variable i as

$$d^{(i)}_1(\varkappa_i) \equiv \min_{\text{floating}\setminus i} d_1(\boldsymbol{\varkappa}_{\text{floating}}|\boldsymbol{\varkappa}_{\text{locked}})$$

where the min is taken over all floating variables other than i and where floating variable i is locked to value $\varkappa_i$.

Since each continuous/range/discrete distance contributes additively to the total distance, we can rank the possible values for any given floating variable i according to $Z^{(i)}_{\text{l,f}}(\varkappa_i) + d^{(i)}_1(v_i)$. The best (according to this heuristic) setting for floating variable i is

$$\varkappa^*_i = \arg\min_{\varkappa_i} \Big\{Z^{(i)}_{\text{l,f}}(\varkappa_i) + d^{(i)}_1(v_i)\Big\}.$$

The heuristic then suggests a variable ordering in order of increasing $Z^{(i)}_{\text{l,f}}(\varkappa^*_i) + d^{(i)}_1(v^*_i)$ so that we first select the floating variable with the lowest sum and last select the variable with the highest sum.

Finally, we turn to the lower bound on the distance. Noting that the continuous and range distance contribution does not have the form of a sum over the dimensions (as the discrete

contribution does), we write the lower bound, lb, as[34]

$$\mathrm{lb} = Z_{\mathrm{l,l}} + \sum_i \left(Z^{(i)}_{\mathrm{l,f}}(\varkappa^*_i) + \delta_{i,i^*} d^{(i)}_1(\varkappa^*_i) \right) + Z^*_{\mathrm{f,f}}$$

where $i^* = \arg\min_i \left(Z^{(i)}_{\mathrm{l,f}}(\varkappa^*_i) + d^{(i)}_1(v^*_i) \right)$ (i.e., i^* labels the floating dimension with the least $Z^{(i)}_{\mathrm{l,f}} + d^{(i)}_1$ contribution to the distance).

5 **E.2.2 Improvements**

We point out that an extensive literature exists on parallel branch and bound algorithms. Many of the techniques developed are applicable to the present problem.

Approved for Public Release; Distribution Unlimited

[34] $\delta_{x,x'}$ is the Kronecker delta function which is 1 if and only if $x = x'$.

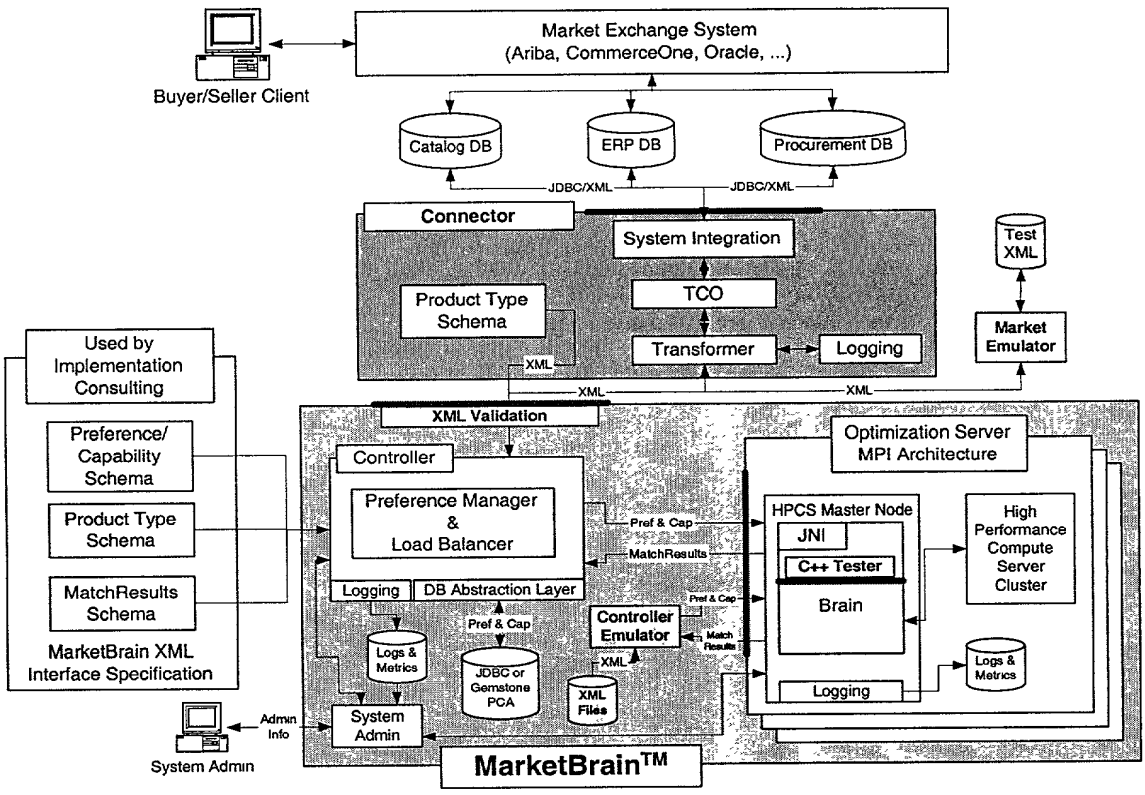


Figure 1: An architecture for the invention.

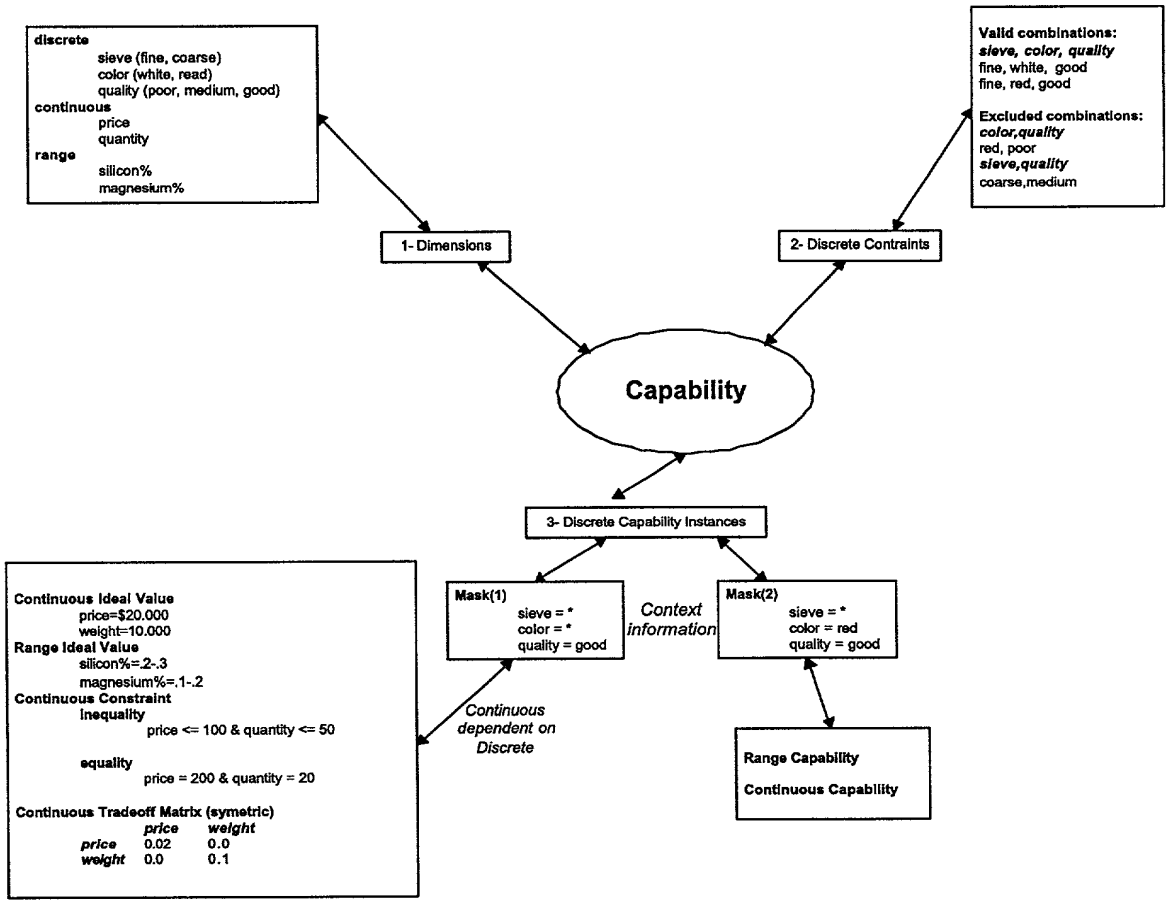


Figure 2: Schematic of a buyer-specific capability with examples indicating potential input.

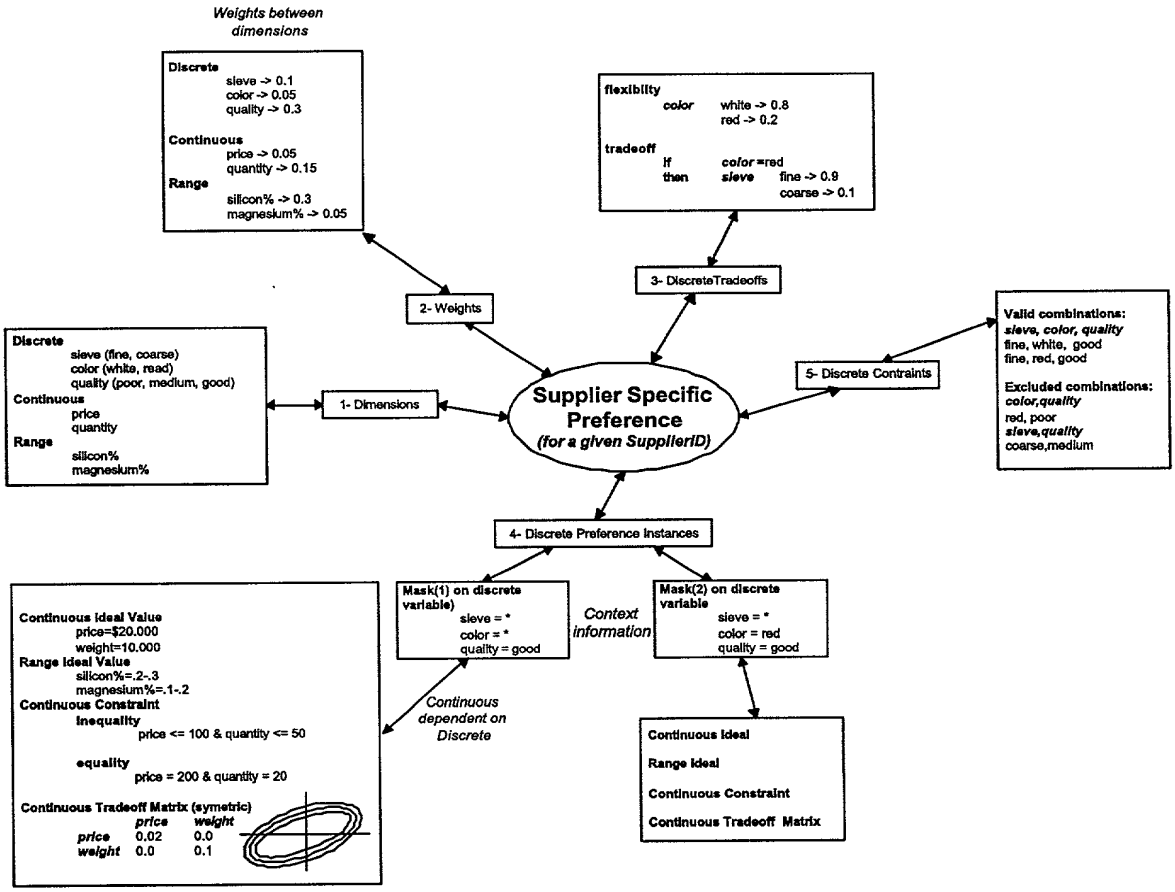


Figure 3: Schematic of a supplier-specific preference with examples indicating potential input.

OIPE
MAY 0 3 2001
JC165
PATENT & TRADEMARK OFFICE

REMARKS

This preliminary amendment replaces identified material incorporated by reference from provisional application no. 60/168,757 (filed on December 6, 1999 and titled, "An E-Commerce Infrastructure for Value Chains") with the actual text. Accordingly, this preliminary amendment does not add new matter. *See e.g.* MPEP § 2163.07(b).

This preliminary amendment also adds 139 new claims. A fee is believed to be due for the additional claims as noted on the attached separate sheet. Should any additional fees be required, please charge them to Pennie & Edmonds LLP Deposit Account No. 16-1150. A copy of this sheet is enclosed. .

Respectfully submitted,

Date: May 3, 2001

Gregory J. Gonsalves, Reg. No. 43,639
For: Francis E. Morris, Reg. No. 24,615

PENNIE & EDMONDS LLP
1667 K Street, N.W.
Washington, D.C. 20006

(202) 496-4400

09/836,250

OIPE JC165
MAY 0 3 2001
PATENT & TRADEMARK OFFICE

Appendix A

Marked-up Version of the Specification

H:\DOCS\CD\CD-3505.DOC

OIPE JC165
MAY 0 3 2001
PATENT & TRADEMARK OFFICE

A Method and System for Discovery of Trades Between Parties

1 Related Applications

This application claims priority to provisional application no. 60/168,754 filed on December 6, 1999, titled, "An E-Commerce Infrastructure for Value Chains", the contents of which are herein incorporated by reference. This application also claims priority to provisional application no. 60/194,880, titled, "Method and System to Mediate Commerce", filed on April 6, 2000, the contents of which are herein incorporated by reference.

H3C5C5 "60/5C5C5"

2 Field of the Invention

The invention relates to a method and system for discovery of trades between parties. In particular, the invention is a system which allows buyers to define their preferences and sellers to define their capabilities, then determines which trading points maximize the utility of the buyer. The system suggests trades by exploiting the flexibilities and tradeoffs encoded by both parties, thus providing win-win trades. A second level of optimization ranks the trades with all suppliers, allowing the buyer to rapidly determine the best alternatives. The system allows for rich negotiation spaces and supports continuous, discrete, and range or interval decision factors.

3 Background of the Invention

The present invention relates to methods of automatic exploration and exploitation of the flexibilities possessed by negotiating parties to uncover improved win-win agreements. The invention describes computationally efficient mechanisms that are applicable whether there are one or many selling parties. The precise number and types of negotiating dimensions are irrelevant as long as they are numerical. Thus the present invention applies equally to the optimal determination of terms in the purchase of a commodity or an arbitrarily complex artifact.

The past 5-10 years have seen remarkable growth in software tools to help firms with enterprise-wide planning (ERP software) and supply chain management (SCM software). While these tools do a wonderful job at integrating disparate data sources within and between firms, the opportunity exists for significant further cost reductions.

The same time period has also seen a tremendous rise in the widespread use of the internet by both consumers and businesses. Forecasters are predicting that within a few years e-commerce between businesses (B2B) and between consumers and businesses (B2C) will grow to in excess of a trillion dollars per year in annual revenues.

Electronic markets have proliferated over the last few years with the advent of B2C (business-to-consumer) and B2B (business-to-business) electronic commerce. Such market places have yielded significant cost savings by lowering the transaction costs between buyers and sellers. Buyers have also profited through increased competition between suppliers. However, electronic markets have hurt suppliers, since the zero-sum negotiation over price has been at their expense. The present invention describes a tool whereby cost savings for both parties are derived from the discovery of win-win trades. Fundamentally, the system works by allowing trading parties to describe their desired trade across multiple dimensions and to express their flexibility around this ideal trade. Through an algorithmic exploration of their flexibilities, the present invention can discover trades that are near the ideal trades of both parties, enabling both to win.

The adoption of B2B and B2C electronic commerce was facilitated by the migration of catalogues online. This familiar method of presentation ameliorated the significant cultural change to electronic trade. For the foreseeable future, electronic commerce will be dominated by online catalogs. At present, online catalogues are direct translations of their hardcopy counterparts where the attributes of a product are described and a price quoted. Inevitably however, online catalogs will become more expressive. Catalog entries will be able to represent price breaks for large quantity orders, lot sizes, etc. Thus it is important that any software (like the present invention) that uncovers mutually beneficial trading scenarios is able to operate with such catalogs. Consequently, in the present invention there is an asymmetry between buyer (usually a human) and seller (usually an online catalog).

One of the reasons catalogs have come to dominate electronic commerce is that the types of goods that can be represented in catalogs are simple. Whether the product is pens or paper clips, different vendor's offers differ little from each other (a pen is a pen is a pen), and a quick scan of a catalog gives a buyer enough information to make an informed purchase. These types of goods are low margin and inexpensive. In contrast, the vast amount of purchasing between businesses involves materials which are directly connected with business operations - car parts, turbines, etc. Such direct goods are the future of electronic commerce. Unlike present-day engines, any truly useful procurement tool must be able to support direct materials with complex attributes and complex inter-relationships between its components.

Electronic commerce offers unprecedented opportunities for more informed decision-making for both buyers and sellers. The past few decades have seen the widespread adoption of enterprise resource planning (ERP) systems, to the point that now almost every major company has some form of ERP software. ERP functions as the digital nervous system of a company, transmitting and logging information between the company's many different business functions. ERP software keeps track of inventory, monitors the state of purchase orders, signals when a company should reorder direct and indirect materials, and a myriad of other functions.

Consequently, ERP databases are a rich source of information to optimize a company's operations. Yet today this information is rarely used to make more informed buying and selling decisions. The present invention can utilize such information sources to optimize a company's interactions with suppliers and customers.

One important manner in which this optimization can occur is through an analysis of *all* cost factors. Current buying and selling practices often focus on limited goals, e.g., minimize the total purchase price. Myopic purchasing strategies often result in higher total cost of ownership when all cost factors relevant to a product in its lifetime of use are included. These other cost factors can be significant. Why save the money in taking delivery two days late if the receiving docks will be full at that time and an additional shift needs to be hired to clear the docks? Why order the cheaper drill bit if it is much more expensive to replace when it breaks? The present invention improves trades by minimizing the total cost of ownership of a product, yielding significant savings to its users. Many total cost factors are difficult to quantify – e.g. what is the cost of dealing with a unionized versus a non-unionized supplier? Consequently, the present invention supports qualitative (best guesses and intuition) as well quantitative factors.

All companies are situated in a supply or value chain. At each step in the chain, a company purchases from its suppliers, transforms these inputs, and sells the output to its customers. The termination of the supply chain is the sale of the final product to the end consumer. Since the only influx of external capital comes from the end consumer, companies have realized that they compete not only as individuals but also as entire supply chains. As result, software products have recently become available which attempt to streamline the operations of links within the entire supply chain. This software, variously called supply chain optimization (SCO) or advanced planning and optimization (APO), operates on the basis of forecasted demand at various points within the supply chain. Based on these predictions, plans are generated telling companies how much to produce and how to schedule their operations. SCO systems are a valuable source of intra-company information – data the present invention capitalizes on. Because SCO software relies on forecasted demand, it is only as helpful as the forecast is accurate, and, unfortunately, in many cases demand is very difficult to predict. How can the software know that laundry detergent will go on special at grocery stores in the Northeast in 7 weeks? As a result of the volatility in demand and the many other unpredictable perturbations that plague supply chains, companies keep significant buffers in the form of inventories. In addition to planning, businesses must also be able to adapt to unplanned effects. Such adaptation requires flexibility and a means to exploit that flexibility. The present invention exploits the flexibility of trading parties to streamline the operations of supply chains by smoothing the boundaries between trading parties.

The present invention is therefore a system to allow trading parties to express trading desires and constraints across many and varied different factors. These trading preferences are informed by many different data sources to optimize for a company's internal operations and its connections to it's supply chain through an analysis including total cost factors. The flexibility expressed by all trading parties is exploited to locate win-win opportunities for all

parties if they exist.

4 Summary of the Invention

We describe the present invention in its application to facilitating trade between buyers and sellers, but note that the mechanisms described are much more general. We can easily imagine, for example, using the present invention to match individuals (with the desires and skills) to projects.

The inspiration for the present invention comes from utility theory developed by economists since the 1960's. Since we are interested in multiple dimensions of negotiation, we draw from the multi-attribute utility theory literature.[1] Utility is an abstract concept which has been formalized in various ways. For the present purposes utility, u, is a number between 0 and 1 representing a party's willingness to trade. Larger values indicate a greater willingness.

4.1 The Negotiation Space

In any negotiation the parties must come to agreement on the factors requiring negotiation. We call these factors dimensions or variables. As an example, when purchasing a car, the buyer may be concerned with price, time of delivery, and color. Each factor price, time, and color is a dimension. Most dimensions can be classed as one of three types: continuous, discrete, or range/interval. A continuous dimension is one like price for which the buyer's utility varies smoothly across that dimension. The buyer's utility at \$23 001.00 is almost the same as the utility at \$23 000. Color is a discrete dimension. Since the car may only be available in black, white, and silver, the domain of this dimension is the finite set of values {black, white, silver}. Moreover, the buyer's utility may be quite different for the three colors. The third class of dimensions is called interval dimensions. An interval dimension arises often in B2B negotiations. If a machined part is built to some tolerance (e.g., the inner diameter of a screw is between 24.5 and 25.5 mm), the range of variability in the dimension is specified as an interval. In the language of statistical quality control, a certain percentage of the machined parts will fall in this range. These three broad classes of variables capture almost all the types of attributes relevant to B2B negotiation.

The present invention operates over any number of continuous, discrete, and range or interval variables. We call the negotiation space X and any point in the negotiation space $(\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r}) \in X$. It is important to recognize that the single trading point $(\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r})$ may have multiple components, e.g., price = \$23 000, time of delivery = 3 weeks, color = black.

In the present invention, the space of negotiation is agreed upon by all parties involved prior to the commencement of any negotiation. We can, however, imagine more dynamic situations in which dimensions are introduced and discarded over time.

[1] For a good introduction to multi-attribute utility theory see [1].

4.2 The Buyer's Utility Function

A party defines it's utility function over this space so that every $(\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r})$ is assigned a utility number indicating the party's willingness to trade. We indicate the utility function as $u((\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r}))$. A great deal of work has been done on the appropriate form for utility functions. In the present invention, we take a simple form for the utility function for two reasons. First, we would like the form of the utility to be conducive to rapid computation. Second we would like the utility to be simple enough to be easily understood by and elicited from users of the invention. With no loss in generality, we write the utility function as $u((\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r})) = \exp(-d((\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r})))$ where $d(\mathbf{x})$ is interpreted as the distance of trading point $(\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r})$ from the most preferred trade.

So that we can operate against seller catalogs, only the buyer needs to define a utility function. Across the continuous dimensions, the buyer's utility is defined by specifying the most preferred (or ideal) continuous dimensions and the manner in which utility drops off as we move away from this ideal. For the discrete dimensions, the utility is specified in tabular form since there are a finite number of alternatives. Again, the buyer must specify it's ideal discrete values and how utility decays away from those values. In section 6.1 we describe how this is accomplished. The range dimensions contribute to utility similarly; the buyer specifies an ideal range and the utility decays for ranges other than the ideal according to their distance from the ideal.

The utility function can also express tradeoffs between variables, e.g., I may take delivery in 5 weeks if the price drops to $20 000, or I may accept the white car if I can take delivery in 2 weeks. The tradeoffs may be between pairs of continuous dimensions (as in the first case), between pairs of discrete variables, or between continuous and discrete variables (as in the second case).

4.2.1 Normalization and Weighting

When utility is defined over different types of variables, it is important to normalize the contributions of each variable so that the buyer can weight the importance of the various contributions to utility. This is a difficult problem. How should a buyer's color preferences be normalized so that they can be traded off against time of delivery? The present invention solves this problem by requiring that the average distance of any negotiation variable from its ideal value is the same for all dimensions. Since the buyer is more interested in those regions of the negotiation space where the utility is high, the average is weighted by utility. This procedure defines a manner in which to define a baseline where all dimensions contribute equally. Given this baseline, the buyer can then weight the various contributions and obtain useful results.

4.2.2 Utility Elicitation

Since utility is fundamental to the present invention, its elicitation from the buyer is important. Utility may be defined using any of a number of sources:

1. graphical user interfaces associated with the invention
2. standard benchmark criteria applicable to the domain
3. formal methodologies like the analytical hierarchical process [2], or discrete choice analysis [3]
4. inferred through models

We expand briefly upon method 4. As discussed in the background section, it is important to buyers to minimize their total cost of ownership. If we have a function representing these costs as a function of the negotiation variables, and perhaps other factors, this function can be used to infer a utility function which will act to minimize the total costs. Later we describe how this can be accomplished.

4.3 A Supplier's Capabilities

As noted previously, suppliers are treated differently from buyers so that the tool can operate against catalog information with no human intervention required on the part of the seller. In fact, we do not require sellers to define a utility at all.

A supplier cannot offer deals at all points within the negotiation space X, e.g., he certainly can't offer the black car tomorrow for free! A capability then represents the ability of a supplier to deliver and defines a subspace of X. It can include such things as price discounts on large volume orders, variation in delivery time as a function of price, etc. Since these relationships are already specified by businesses in terms of simple rules like *"the price per unit is $10.00 if 1 to 999 units are ordered and $9.50 per unit if 1000 or more units are ordered"*, suppliers' capabilities are represented in the present invention by piecewise linear functions.

4.4 Negotiation Constraints

Both parties may have constraints which must be satisfied in order for them to trade. For example, the buyer may not buy the car unless he gets it within 6 weeks, or he may not purchase the car if it is available only in white. These are examples of continuous and discrete constraints, respectively. A continuous constraint sets a requirement on the continuous variables. In the present invention, continuous constraints must be either linear or quadratic. Discrete constraints involve discrete variables. A discrete constraint can be expressed as a list of the allowed (or disallowed) combinations of the discrete variables for which the trade will be acceptable. For example, if the buyer would accept either the black or the silver car, the constraint would list both these colors as viable. It is important to note that both continuous and discrete constraints may involve one or more variables. We can also express constraints involving both types of variables by allowing the continuous constraints to differ depending on the discrete variables.

4.5 Utility Optimization

With the major components of the invention in place, we describe how the overall system works. As a procurement tool for the buyer, there are two levels of optimization. First, for any given supplier we maximize the buyer's utility, subject to the supplier's capabilities to find that trade which makes the buyer as happy as possible. Since we are optimizing within a supplier's capabilities, the supplier has expressed a willingness to complete the trade at whatever point is determined to be optimal. The tool then optimizes across suppliers to rank them according to utility at the optimal point. A graphical user interface allows a buyer to investigate the trades suggested by the tool by sorting according to any dimension or by the overall utility.

Utility, while a useful concept in assessing an overall score, may be of limited use to a buyer due to its abstract meaning. Consequently, we can also apply the total cost of ownership function to the results to rank order the suggested trades according to their various cost components. Recall that for any trade $\mathbf{x} \in X$, the total cost of ownership function returns the various cost contributions. This additional information aids the buyer in his purchasing decision. The utility number for each trade is still useful because the total cost of purchase function includes only those cost factors which can be quantified, whereas the utility also includes "softer" qualitative factors.

4.5.1 Aggregation

In addition to optimizing against one supplier at a time, the present invention can also be used to optimize against an arbitrary aggregation of suppliers. This is important if, for example, no single seller can supply the large volume requested by a buyer. In this mode of operation, the buyer specifies sets of suppliers participating in the aggregation and the dimensions over which aggregation can occur, and the tool finds the optimal combination in which to distribute the volume dimension over the allowed suppliers.

4.6 An e-Commerce Infrastructure for Value Chains

This patent application also describes an integrated solution for B2C and B2B e-commerce that would be built on top of ERP and SCM software and which would provide a number of compelling benefits to companies. Amongst the benefits are:

- multidimensional markets which allow consumers to implicitly define their preferences over many criteria. This allows both consumers to express what it is they really value, allows companies to position themselves clearly in the space of value, and allows for efficient matches between trading partners
- optional anonymity of market participants and their trading desires when that is appropriate[2]
- explicit pricing of the flexibility possessed by the consumer and all businesses in the supply chain which allows for more robust operation of the entire supply chain. This

[2] Anonymity is not, however, a pre-requisite of the proposed invention.

concept is very different from other types of markets (*e.g.* auctions, reverse auctions, exchanges) where transactions are specified exactly. The flexibility introduced by any party, whether consumer or supplier, is propagated and exploited through the entire supply chain.

- capture and quantification of true consumer demand leading to improved forecasting and product development by suppliers
- automated markets that integrate supply chain networks through coordination across and within company boundaries. Coupling of the automated markets with local (*i.e.* at the company level) optimization tools fed by real-time company data allows for optimization and cost savings across the entire supply chain.

It should be recognized that supply chains may be very different in the near future. Current supply chains are based on physical objects made valuable through a sequence of transformations resulting in a product purchased by an end consumer. With the move to an information economy the supply chain of the near future may not involve physical goods at all. In particular the entire supply chain may consist of value adding operations converting raw data to consumer-desired information. Such supply chains will have the same coordination problems current ones do. Our proposed solution applies equally well to these future supply chains and by supply chain we mean this more general notion.

5 Brief Description of the Figures

Figure 1 shows an [an] architecture for the invention.

Figure 2 shows a schematic of a buyer-specific capability with examples indicating potential input.

Figure 3 shows a schematic of a supplier-specific preference with examples indicating potential input.

6 Detailed Description

6.1 Theory

In this section we outline the mathematical foundations of the optimization process in sufficient detail to allow for computer implementation.

6.1.1 The Negotiation Space

In Table 1 we define the parameters which collectively define the space of negotiation X. For each of the n_c continuous variables, we specify an allowed range over which that continuous dimension may vary as $x_i \in X_i = [\underline{x}_i, \overline{x}_i]$, where $\underline{\mathbf{x}}$ is the n_c-vector of lower continuous bounds and $\overline{\mathbf{x}}$ is the n_c-vector of upper continuous bounds. Each discrete variable assumes a value from within its domain $\varkappa_i \in \mathcal{D}_i$. Without loss of generality, we label the domain of discrete

n_c	number of continuous dimensions
n_d	number of discrete dimensions
n_r	number of range dimensions
$\mathbf{x}$	n_c-vector of values for continuous dimensions
$\boldsymbol{\varkappa}$	n_d-vector of values for continuous dimensions
x_i	value of ith continuous variable
$\varkappa_i$	value of ith discrete variable

Table 1: Definition of the negotiation search space.

variable i by $\mathcal{D}_i = [1, \cdots, d_i]$ where $d_i \geq 0$ is an integer giving the number of possible values discrete variable $\varkappa_i$ may assume.

With these definitions, we define the space of negotiation by the tensor product $X = X_1 \otimes \cdots \otimes X_{n_c} \otimes \mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_{n_d}$. Range variables are treated separately and not negotiated over.

6.1.2 The Utility Function

The utility function is a mapping from X into the interval $[0, 1]$. As indicated earlier we assume the utility to have the form $u(\mathbf{x}, \boldsymbol{\varkappa}) = \exp[-d(\mathbf{x}, \boldsymbol{\varkappa})]$ where $d(\mathbf{x}, \boldsymbol{\varkappa})$ is interpreted as a distance. In what follows we will assume that in its simplest form the distance function has the form

$$d(\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r}) = \big(\mathbf{x} - \boldsymbol{\mu}(\boldsymbol{\varkappa})\big)^t \mathbf{C}^{-1}(\boldsymbol{\varkappa}) \big(\mathbf{x} - \boldsymbol{\mu}(\boldsymbol{\varkappa})\big) + Z(\boldsymbol{\varkappa}) + R(\mathbf{r}; \boldsymbol{\varkappa}).$$

Each contribution to the distance function is positive. We consider each contribution to the distance in turn, beginning with the range variable contribution $R(\mathbf{r}; \boldsymbol{\varkappa})$.

First, we note that the range distance depends on the setting of the discrete variables. This allows the buyer to express different preferences for the range variables depending on discrete factors. The total range distance is summed up over all possible range variables so that $R(\mathbf{r}; \boldsymbol{\varkappa}) = \sum_{i=1}^{n_r} R_i(r_i; \boldsymbol{\varkappa})$. The vector $\mathbf{r}$ indicates the preferred values for all range variables. If range variable i is specified as the interval $r_i \equiv (\underline{r}_i, \overline{r}_i)$ (where $\overline{r}_i > \underline{r}_i$) then $\mathbf{r}$ is an n_r-vector of such tuples. The distance contribution, R_i, from one range variable will depend on the application. If the range variables are meant to represent the tolerances on machined parts where issues of statistical quality control are important, then the distance between two ranges might be related to the overlap between Gaussian distributions. If r_i is interpreted as a Gaussian having mean $(\underline{r}_i + \overline{r}_i)/2$ and standard deviation proportional to $\overline{r}_i - \underline{r}_i$ then an appropriate range distance is given in Appendix A. Other choices for the range distance function are certainly possible.

The continuous distance is quadratic and determined by the positive semidefinite $n_c \times n_c$ matrix $\mathbf{C}^{-1}$. We have allowed this matrix to vary with the setting of the discrete variables and indicated this explicitly through $\mathbf{C}^{-1}(\boldsymbol{\varkappa})$.[3] The n_c-vector $\boldsymbol{\mu}$ may also depend on $\boldsymbol{\varkappa}$ and indicates the point at which the utility is maximal – $\boldsymbol{\mu}$ is thus identified with the ideal value for the continuous variables. The precise quadratic form is convenient, but, using recent developments

[3]Where no confusion will arise we eliminate the dependence of $\mathbf{C}^{-1}$ on $\boldsymbol{\varkappa}$ for notational simplicity.

in interior point methods, other convex functions are also computationally tractable [4].

The discrete distance is determined through the function $Z(\boldsymbol{\varkappa})$ which maps the discrete space $\mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_{n_d}$ onto the positive real line $[0, \infty]$. In keeping with the assumption that distance is a function of only pairs of components x_i, x_j, we assume the discrete distance has the form[4]

$$Z(\boldsymbol{\varkappa}) = \sum_{i=1}^{n_d} \Big\{ Z_i(\varkappa_i) + \sum_{j=1(\neq i)}^{n_d} Z_{i,j}(\varkappa_i, \varkappa_j) \Big\}.$$

Each contribution $Z_{i,j}$ is a table consisting of $d_i d_j$ entries, where $Z_{i,j}(\varkappa_i, \varkappa_j)$ can be interpreted as the distance if discrete dimension i has value $\varkappa_i$ conditioned on discrete dimension j having value $\varkappa_j$. The diagonal terms Z_i offer an unconditional distance. The most preferred value for the ith discrete dimension is that for which $Z_i(\varkappa_i) = 0$.

Rather than require the user to enter the distances explicitly, there are numerous ways in which the distances can be generated automatically based upon a buyer's ranking of preferred values. Further details can be found in Appendix B.

Weighting of Dimensions

In many cases it is important for simple modifications of the distance function to re-weight the contributions to the total distance. If $\mathbf{w}_c$ is an n_c-vector of weights for the continuous dimensions, we can accomplish this by letting $\mathbf{C}_w^{-1} = \mathbf{W}_c \mathbf{C}^{-1} \mathbf{W}_c$ where $\mathbf{W}_c$ is the diagonal matrix $\mathbf{W}_c = \mathrm{diag}(\mathbf{w}_c)$.[5] In a similar way we modify the discrete distance to $Z_{w;i,j}(\varkappa_i, \varkappa_j) = w_{d;i} w_{d;j} Z_{i,j}(\varkappa_i, \varkappa_j)$ where $\mathbf{w}_d$ is the n_d-vector of weights for the discrete variables and $w_{d;i}$ is its ith component. The range contribution is also modified so that $R_{w;i}(r_i) = w_{r;i} R_i(r_i)$ where $\mathbf{w}_r$ is the n_r-vector of weights for the range variables and $w_{r;i}$ is its ith component. For convenience the weights are normalized so that $(\mathbf{1}^t \mathbf{w}_c)^2 + (\mathbf{1}^t \mathbf{w}_d)^2 + \mathbf{1}^t \mathbf{w}_r = 1$. With little additional complexity the dimension weights can be made dependent on the setting of the discrete variables but we will assume throughout that the weights are constant.

With these modifications, the total distance function becomes

$$d(\mathbf{x}, \boldsymbol{\varkappa}) = \big(\mathbf{x} - \boldsymbol{\mu}(\boldsymbol{\varkappa})\big)^t \mathbf{C}_w^{-1}(\boldsymbol{\varkappa}) \big(\mathbf{x} - \boldsymbol{\mu}(\boldsymbol{\varkappa})\big) + Z_w(\boldsymbol{\varkappa}) + R_w(\mathbf{r}). \tag{1}$$

where $Z_w(\boldsymbol{\varkappa}) = \sum_{i=1}^{n_d} w_{d;i} \big\{ w_{d;i} Z_i(\varkappa_i) + \sum_{j=1(\neq i)}^{n_d} w_{d;j} Z_{i,j}(\varkappa_i, \varkappa_j) \big\}$ and $R_w(\mathbf{r}) = \sum_{i=1}^{n_r} w_{r;i} R_i(r_i)$

Assigning weighting factors is useful only if the relevant contributions have been previously normalized so that they are all roughly the same magnitude. This serves as the baseline for which all weights are equal. The question immediately arises as to what criteria to use to weight the distance contributions.

We shall determine scaling factors, $Q_d > 0$ and $Q_r > 0$, so that the average distances per dimension of the discrete, range, and continuous contributions are equal, where by average we mean a utility-weighted average over the entire space of possible trades. This weighting places more emphasis on the better trades

If d_c, d_d, and d_r are the continuous, discrete, and range contributions to the total distance, then after multiplication by the scaling factors $d = d_c + Q_d d_d + Q_r d_r$. The scaling factors are

[4] Later we shall generalize this distance to include weighting of dimensions.

[5] $\mathbf{M} = [m_{i,j}] = \mathrm{diag}(\mathbf{v})$ is the diagonal matrix formed by setting $m_{i,i} = v_i$ and $m_{i,j} = 0$ for $j \neq i$.

determined through the utility weighted average distances defined by

$$\langle d_r \rangle \equiv \frac{\sum_{\boldsymbol{\varkappa}} \sum_r \int_V d\mathbf{u}\, Q_r d_r \exp[-Q_r d_r - Q_d d_d - d_c]}{\sum_{\boldsymbol{\varkappa}} \sum_r \int_V d\mathbf{u}\, \exp[-Q_r d_r - Q_d d_d - d_c]} = Q_r \frac{\sum_{\boldsymbol{\varkappa}} \exp[-Q_d d_d] \sum_r d_r \exp[-Q_r d_r] \int_V d\mathbf{u}\, \exp[-d_c]}{\sum_{\boldsymbol{\varkappa}} \exp[-Q_d d_d] \sum_r \exp[-Q_r d_r] \int_V d\mathbf{u}\, \exp[-d_c]}$$

$$\langle d_d \rangle \equiv \frac{\sum_{\boldsymbol{\varkappa}} \sum_r \int_V d\mathbf{u}\, Q_d d_d \exp[-Q_r d_r - Q_d d_d - d_c]}{\sum_{\boldsymbol{\varkappa}} \sum_r \int_V d\mathbf{u}\, \exp[-Q_r d_r - Q_d d_d - d_c]} = Q_d \frac{\sum_{\boldsymbol{\varkappa}} d_d \exp[-Q_d d_d] \sum_r \exp[-Q_r d_r] \int_V d\mathbf{u}\, \exp[-d_c]}{\sum_{\boldsymbol{\varkappa}} \exp[-Q_d d_d] \sum_r \exp[-Q_r d_r] \int_V d\mathbf{u}\, \exp[-d_c]}$$

$$\langle d_c \rangle \equiv \frac{\sum_{\boldsymbol{\varkappa}} \sum_r \int_V d\mathbf{u}\, d_c \exp[-Q_r d_r - Q_d d_d - d_c]}{\sum_{\mathbf{v}} \sum_r \int_V d\mathbf{u}\, \exp[-Q_r d_r - Q_d d_d - d_c]} = \frac{\sum_{\boldsymbol{\varkappa}} \exp[-Q_d d_d] \sum_r \exp[-Q_r d_r] \int_V d\mathbf{u}\, d_c \exp[-d_c]}{\sum_{\boldsymbol{\varkappa}} \exp[-Q_d d_d] \sum_r \exp[-Q_r d_r] \int_V d\mathbf{u}\, \exp[-d_c]}$$

A few comments on the above equations are in order. First, $\sum_{\boldsymbol{\varkappa}}$ indicates the repeated sum $\sum_{\varkappa_1} \cdots \sum_{\varkappa_{n_d}}$ over all possible discrete trades. $\sum_r$ indicates a sum over all the range variables and the integral over volume V indicates integration over the continuous trading volume of interest. Finally, we have not included a scaling factor Q_c on the continuous distance, since this can be made equal to 1 if we reinterpret Q_r as Q_r/Q_c and Q_d as Q_d/Q_c.[6] Each of the averages is an explicit function Q_d and Q_r.

The requirement on equal average contributions determines the two unknowns Q_r and Q_d through the equations: $\langle d_r \rangle / n_r = \langle d_c \rangle / n_c$ and $\langle d_d \rangle / n_d = \langle d_c \rangle / n_c$. These two nonlinear equations are coupled in terms of Q_r and Q_d and must be solved simultaneously for Q_r and Q_d. Further details are found in Appendix C.

6.1.3 Constraint Specification

Buyers and sellers may express constraints over both continuous and discrete variables.

Continuous Constraints

For simplicity (and because additional expressiveness is rarely required) we assume that the buyer's constraints over the continuous variables are linear.[7] This allows a buyer to express a constraint, e.g., the time of delivery must be within 10 days or I will not trade, i.e., $t \leq 10$. We allow for both inequality and equality constraints which can be expressed as $\mathbf{G}_1^{(b)}\mathbf{x} = \mathbf{g}_1^{(b)}$ and $\mathbf{G}_2^{(b)}\mathbf{x} \leq \mathbf{g}_2^{(b)}$. If there are $c_1^{(b)}$ equality constraints then $\mathbf{G}_1^{(b)}$ has $c_1^{(b)}$ rows. Similarly, $\mathbf{G}_2^{(b)}$ has $c_2^{(b)}$ rows if there are $c_2^{(b)}$ inequality constraints. We allow the constraints to depend on the setting of the discrete variables, and to be explicit we often write $\mathbf{G}_1^{(b)}(\boldsymbol{\varkappa})$, $\mathbf{g}_1^{(b)}(\boldsymbol{\varkappa})$, $\mathbf{G}_2^{(b)}(\boldsymbol{\varkappa})$, and $\mathbf{g}_2^{(b)}(\boldsymbol{\varkappa})$.

Discrete Constraints

We use a standard methodology to represent and process constraints over discrete variables [5]. Abstractly, a constraint over a (perhaps proper)[8] subset of the discrete variables is represented as a list of all the allowed combinations the variables may assume. An example representation of a pair of discrete constraints is given in Table 2. There are two solutions to this set of constraints: $\varkappa_1 = 1$, $\varkappa_2 = 2$, $\varkappa_3 = 3$ and $\varkappa_1 = 3$, $\varkappa_2 = 2$, $\varkappa_3 = 1$. We indicate these solutions as $\{\langle \varkappa_1, 1\rangle, \langle \varkappa_2, 2\rangle\}, \langle \varkappa_3, 3\rangle\}\}$ and $\{\langle \varkappa_1, 3\rangle, \langle \varkappa_2, 2\rangle\}, \langle \varkappa_3, 1\rangle\}\}$ respectively. Each solution where all the variables have been identified with specific values from their domains is called a labelling.

Computationally efficient representations are used to ensure that only feasible combinations

[6] We singled the continuous variables out, since there will always be continuous variables in any trading scenario.

[7] With little additional complexity we can also handle quadratic constraints, see [4].

[8] A proper subset of a set is the set itself.

$\varkappa_1$	$\varkappa_2$	$\varkappa_3$
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

(a)

$\varkappa_2$
2

(b)

Table 2: An example set of constraints involving 3 variables where the domains of all variables are $\mathcal{D}_1 = \mathcal{D}_2 = \mathcal{D}_3 = \{1, 2, 3\}$. Constraint (a) requires that the values assumed by $\varkappa_1$, $\varkappa_2$, and $\varkappa_3$ are all different from each other, and constraint (b) requires that the value assumed by $\varkappa_2$ is even. See text for the solution to both constraints.

of variables are ever processed. Numerous third-party libraries offer constraint programming functionality.[9]

6.1.4 Utility and Total Cost of Ownership

The buyer's utility function and associated constraints may be difficult for many users to define. In this section we show how models of the buyer's business can be used to define utility in a natural manner.

We imagine a function which provides an estimate of the total cost of ownership for any given purchase. Cost contributions to this function might include piece part costs, freight costs, setup costs, quality assurance costs, repair costs, etc. It is important to include all quantifiable costs associated with the lifetime of use of the purchased product because it is this function we will be minimizing. Significant savings may be obtained by taking a longer-term view of the purchase. Revenues (negative costs) generated from the purchase are also included in the function so that the function represents some measure of profitability associated with the purchase. We write the total cost of ownership function as $C_o(\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r}; \boldsymbol{\beta})$. We explicitly indicate the dependence on the negotiated trade parameters $\mathbf{x}$, $\boldsymbol{\varkappa}$, and $\mathbf{r}$, as well as other factors $\boldsymbol{\beta}$. The other factors might include forecasted demand, current inventory levels, etc. These factors will vary over time, and they can be extracted from the buyer's ERP and supply chain management systems (SCM) in real-time just before the purchase to ensure continuous real-time optimization. See section 6.2.1 for further details.

Minimization of $C_o(\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r}; \boldsymbol{\beta})$ defines an ideal trade dependent on current conditions: $\mathbf{x}_{\text{opt}}(\boldsymbol{\beta})$, $\boldsymbol{\varkappa}_{\text{opt}}(\boldsymbol{\beta})$, $\mathbf{r}_{\text{opt}}(\boldsymbol{\beta})$. If desired, these can be used to define $\boldsymbol{\mu} = \mathbf{x}_{\text{opt}}(\boldsymbol{\beta})$, $r = \mathbf{r}_{\text{opt}}(\boldsymbol{\beta})$ and the desired ideal discrete configuration $\boldsymbol{\varkappa}_{\text{opt}}(\boldsymbol{\beta})$ (having distance contribution $Z = 0$). Moreover, the tradeoffs between continuous dimensions around this minimum can be obtained through calculation of the Hessian matrix $\mathbf{H} = [h_{i,j}]$ where the i, j matrix element is given by

$$h_{i,j} = \left. \frac{\partial^2 C_o(\mathbf{x}, \boldsymbol{\varkappa}, \mathbf{r}; \boldsymbol{\beta})}{\partial x_i \partial x_j} \right|_{\mathbf{x}=\mathbf{x}_{\text{opt}}(\boldsymbol{\beta}), \boldsymbol{\varkappa}=\boldsymbol{\varkappa}_{\text{opt}}(\boldsymbol{\beta}), \mathbf{r}=\mathbf{r}_{\text{opt}}(\boldsymbol{\beta})}$$

[9] See for example `www.mozart-oz.org` or `www.ilog.com`.

We then identify $\mathbf{C}^{-1}$ with $\mathbf{H}$. In this way, little trading flexibility is obtained in directions where total cost of ownership rises rapidly, while significant flexibility is obtained in directions where total cost of ownership increases slowly.

In summary, a total cost of ownership model defines both the most preferred trade parameters and the flexibility possessed around the preferred trade. The model pulls dynamically from real-time data sources to provide the most up-to-date optimization based on total costs of ownership and other important qualitative factors the buyer may wish to describe in the utility function. The same function and its constituent costs may also be used to help analyze proposed trades from suppliers.

6.1.5 Supplier Capabilities

As discussed in the introduction, suppliers represent their capabilities through a specification of the subspace of X in which they will trade. A supplier's capabilities must specify the allowed continuous, discrete, and range variables. The allowed range variables are expressed as the pairs $(\underline{r}_j, \overline{r}_j)$, one for each range variable. For example, if a supplier produces 25mm inner diameter screws to within a tolerance of 0.5 mm, then the range variable is simply $(24.5, 25.5)$. These are compared with the buyer's ideal range and contribute to the distance function through the $\mathbf{R}_w(\boldsymbol{\varkappa})$ function.

Capabilities over continuous and discrete variables are more complex.

Continuous Capabilities

Continuous capabilities are viewed naturally as responses to a buyer's request. Thus we distinguish between a buyer's requested continuous vector $\mathbf{x}^{(b)}$ and a seller's response $\mathbf{x}^{(s)}$. A vector-valued function, $\mathbf{f}(\mathbf{x}^{(b)}, \mathbf{x}^{(s)}, \boldsymbol{\varkappa})$ returns the response based on the buyer's request and also, perhaps, other previously defined supplier responses. Component f_i of $\mathbf{f}$ defines the ith continuous variable, *i.e.* $x_i^{(s)} = f_i(\mathbf{x}^{(b)}, \mathbf{x}^{(s)})$.

Currently, suppliers are used to quoting price discounts for large volume orders and these price discounts are expressed as piecewise linear functions. Consequently, we restrict f_i to have the following form (where we distinguish between the functions depending on the buyer and seller variables):

$$x_i^{(s)} = \sum_k f_{i,k}^{(s)}(x_k^{(s)}, \boldsymbol{\varkappa}^{(s)}) + f_{i,k}^{(b)}(x_k^{(b)}, \boldsymbol{\varkappa}^{(s)}). \tag{2}$$

An example of how this may be used to define a supplier response is the following: We assume three continuous dimensions – price, volume, and time of delivery and indicate these as $[x_1, x_2, x_3] = [p, v, t]$. Then a response may be formed as

$$\begin{aligned} v^{(s)} &= f_{v,v}(v^{(b)}) \\ t^{(s)} &= f_t(v^{(s)}, t^{(b)}) = f_{t,v}(v^{(s)}) \\ p^{(s)} &= f_p(v^{(s)}, t^{(s)}) = f_{p,v}(v^{(s)}) + f_{p,t}(t^{(s)}). \end{aligned}$$

The $f_{v,v}$ function returns the volume a supplier will fulfill as a function of what the buyer asked for. If the supplier can deliver any volume, this will be the identity function. If the

supplier delivers only in certain lot sizes, this function may have a staircase shape, etc. The $f_{t,v}$ function indicates the time it will take a supplier to deliver a certain volume. So, for example, if larger shipments require longer transportation, then this dependence is given by this function. Finally, we turn to the price determination. In this example the price depends on the quantity $v^{(s)}$ being shipped and the $f_{p,v}$ might represent price discounts for large volume orders. There is also an incremental price contribution based on the time of delivery. If faster delivery is more expensive, this is reflected in $f_{p,t}$.

For a given setting of the discrete variables, each $f^{(s)}_{i,k}(x^{(s)}_k, \boldsymbol{\varkappa}^{(s)})$ and $f^{(b)}_{i,k}(x^{(b)}_k, \boldsymbol{\varkappa}^{(s)})$is a one-dimensional piecewise linear function. Consequently, the functions can be specified by listing the breakpoints. If $f^{(s)}_{i,k}(x^{(s)}_k, \boldsymbol{\varkappa}^{(s)})$ has $\kappa^{(s)}_{i,k}$ breakpoints, then we list these as $\{x^{(s)}_k(b^{(s)}_{i,k} = 1), x^{(s)}_k(b^{(s)}_{i,k} = 2), \cdots, x^{(s)}_k(b^{(s)}_{i,k} = \kappa^{(s)}_{i,k})\}$[10] and function values at these breakpoints are $\{f^{(s)}_{i,k}\big(x^{(s)}_k(b^{(s)}_{i,k} = 1)\big), f^{(s)}_{i,k}\big(x^{(s)}_k(b^{(s)}_{i,k} = 2)\big), \cdots, f^{(s)}_{i,k}\big(x^{(s)}_k(b^{(s)}_{i,k} = \kappa^{(s)}_{i,k})\big)\}$. Similarly, $f^{(b)}_{i,k}$, is a piecewise linear function defined by the $\kappa^{(b)}_{i,k}$ breakpoints $\{x^{(b)}_k(b^{(b)}_{i,k} = 1), x^{(b)}_k(b^{(b)}_{i,k} = 2), \cdots, x^{(b)}_k(b^{(b)}_{i,k} = \kappa^{(b)}_{i,k})\}$ and function values at these breakpoints $\{f^{(b)}_{i,k}\big(x^{(b)}_k(b^{(b)}_{i,k} = 1)\big), f^{(b)}_{i,k}\big(x^{(b)}_k(b^{(b)}_{i,k} = 2)\big), \cdots, f^{(b)}_{i,k}\big(x^{(b)}_k(b^{(b)}_{i,k} = \kappa^{(b)}_{i,k})\big)\}$. The breakpoints are indexed by the integers $b^{(s)}_{i,k}$ and $b^{(b)}_{i,k}$.

An interval is specified by assigning a value $b^{(s)}_{i,k} \in [1, \kappa^{(s)}_{i,k} - 1]$ and $b^{(b)}_{i,k} \in [1, \kappa^{(b)}_{i,k} - 1]$ so that[11]

$$x^{(s)}_k(b^{(s)}_{i,k}) \le x^{(s)}_k \le x^{(s)}_k(b^{(s)}_{i,k} + 1) \ \ \forall\, i \quad \text{and} \quad x^{(b)}_k(b^{(b)}_{i,k}) \le x^{(b)}_k \le x^{(b)}_k(b^{(b)}_{i,k} + 1) \ \ \forall\, i. \tag{3}$$

Within each interval the functions are linear, so we have

$$x^{(s)}_i = c_i(\boldsymbol{\varkappa}^{(s)}, \{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\}) + \sum_k m^{(s)}_{i,k}(\boldsymbol{\varkappa}^{(s)}, b^{(s)}_{i,k}) x^{(s)}_k + m^{(b)}_{i,k}(b^{(b)}_{i,k}) x^{(b)}_k$$

where $c_i(\mathbf{v}^{(s)}, \{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\}) = \sum_k c^{(s)}_{i,k}(\boldsymbol{\varkappa}^{(s)}, b^{(s)}_{i,k}) + c^{(b)}_{i,k}(b^{(b)}_{i,k})$. In the above equations, the intercepts and slopes are given explicitly by

$$c^{(s)}_{i,k}(b^{(s)}_{i,k}) = \frac{x^{(s)}_k(b^{(s)}_{i,k} + 1) f^{(s)}_{i,k}\big(x^{(s)}_k(b_{i,k})\big) - x^{(s)}_k(b_{i,k}) f^{(s)}_{i,k}\big(x^{(s)}_k(b_{i,k} + 1)\big)}{x^{(s)}_k(b^{(s)}_{i,k} + 1) - x^{(s)}_k(b_{i,k})}$$

and

$$m^{(s)}_{i,k}(b^{(s)}_{i,k}) = \frac{f^{(s)}_{i,k}\big(x^{(s)}_k(b^{(s)}_{i,k} + 1)\big) - f^{(s)}_{i,k}\big(x^{(s)}_k(b_{i,k})\big)}{x^{(s)}_k(b^{(s)}_{i,k} + 1) - x^{(s)}_k(b^{(s)}_{i,k})}$$

respectively. An analogous result holds for the $c^{(b)}_{i,k}(b^{(b)}_{i,k})$ and $m^{(b)}_{i,k}(b^{(b)}_{i,k})$.

To eliminate any cyclic dependence on $x^{(s)}_i$ we must impose an ordering on $x^{(s)}_i$ so that $x^{(s)}_i$ can only depend on $x^{(s)}_j$ where $j < i$. Consequently, we can write

$$x^{(s)}_i = c_i(\boldsymbol{\varkappa}^{(s)}, \{b^{(s)}_{i,1}, \cdots, b^{(s)}_{i,i-1}\}, \{b^{(b)}_{i,k}\}) + \sum_{k<i} m^{(s)}_{i,k}(\boldsymbol{\varkappa}^{(s)}, b^{(s)}_{i,k}) x^{(s)}_k + \sum_k m^{(b)}_{i,k}(b^{(b)}_{i,k}) x^{(b)}_k.$$

[10] The breakpoints are assumed to be in increasing order.

[11] Note that many choices for $\{b^{(s)}_{i,k}\}$ or $\{b^{(b)}_{i,k}\}$ would be inconsistent with these constraints. A simple way to fix this is to define a union of all breakpoints (s) and (b), and have all $f_{i,k}$ have the same breakpoints.

Written as a matrix equation, the above becomes

$$\left(\mathbf{I} - \mathbf{M}^{(s)}(\boldsymbol{\varkappa}^{(s)}, \{b_{i,k}^{(s)}\})\right)\mathbf{x}^{(s)} = \mathbf{c}(\boldsymbol{\varkappa}^{(s)}, \{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\}) + \mathbf{M}^{(b)}(\{b_{i,k}^{(b)}\})\mathbf{x}^{(b)}$$

where $\mathbf{c}(\boldsymbol{\varkappa}^{(s)}, \{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\}) = \left[c_1(\boldsymbol{\varkappa}^{(s)}, \{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\}) \quad \cdots \quad c_n(\boldsymbol{\varkappa}^{(s)}, \{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\})\right]^t$,
$\mathbf{x}^{(s)} = \left[x_1^{(s)} \quad \cdots \quad x_{n_c}^{(s)}\right]^t$, $\mathbf{x}^{(b)} = \left[x_1^{(b)} \quad \cdots \quad x_{n_c}^{(b)}\right]^t$, and

$$\mathbf{M}^{(s)}(\boldsymbol{\varkappa}^{(s)}, \{b_{i,k}^{(s)}\}) = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ m_{2,1}^{(s)}(\boldsymbol{\varkappa}^{(s)}, b_{2,1}^{(s)}) & 0 & \cdots & 0 & 0 \\ m_{3,1}^{(s)}(\boldsymbol{\varkappa}^{(s)}, b_{3,1}^{(s)}) & m_{3,2}^{(s)}(\boldsymbol{\varkappa}^{(s)}, b_{3,2}^{(s)}) & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ m_{n,1}^{(s)}(\boldsymbol{\varkappa}^{(s)}, b_{n,1}^{(s)}) & m_{n,2}^{(s)}(\boldsymbol{\varkappa}^{(s)}, b_{n,2}^{(s)}) & \cdots & m_{n,n-1}^{(s)}(\boldsymbol{\varkappa}^{(s)}, b_{n,n-1}^{(s)}) & 0 \end{bmatrix}$$

and

$$\mathbf{M}^{(b)}(\{b_{i,k}^{(b)}\}) = \begin{bmatrix} m_{1,1}^{(b)}(b_{1,1}^{(b)}) & m_{1,2}^{(b)}(b_{1,2}^{(b)}) & \cdots & m_{1,n}^{(b)}(b_{1,n}^{(b)}) \\ m_{2,1}^{(b)}(b_{2,1}^{(b)}) & m_{2,2}^{(b)}(b_{2,2}^{(b)}) & \cdots & m_{2,n}^{(b)}(b_{2,n}^{(b)}) \\ \vdots & \vdots & \ddots & \vdots \\ m_{n,1}^{(b)}(b_{n,1}^{(b)}) & m_{n,2}^{(b)}(b_{n,2}^{(b)}) & \cdots & m_{n,n}^{(b)}(b_{n,n}^{(b)}) \end{bmatrix}.$$

In most cases $\mathbf{x}^{(s)}$ will depend only on a subset of the variables in $\mathbf{x}^{(b)}$. If $\mathbf{x}^{(s)}$ depends on $n' \leq n$ of the $\mathbf{x}^{(b)}$ variables, then $\mathbf{M}^{(b)}$ is an $n \times n'$ matrix. In the example given everything depending only upon the volume the buyer requested.

Since $\mathbf{M}^{(s)}(\boldsymbol{\varkappa}^{(s)})$ is lower triangular and can be inverted in time $\mathcal{O}(n)$, we can rapidly express $\mathbf{x}^{(s)}$ as

$$\mathbf{x}^{(s)} = \left(\mathbf{I} - \mathbf{M}^{(s)}(\boldsymbol{\varkappa}^{(s)}, \{b_{i,k}^{(s)}\})\right)^{-1}\mathbf{c}(\boldsymbol{\varkappa}^{(s)}, \{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\}) + \left(\mathbf{I} - \mathbf{M}^{(s)}(\boldsymbol{\varkappa}^{(s)}, \{b_{i,k}^{(s)}\})\right)^{-1}\mathbf{M}^{(b)}(\{b_{i,k}^{(b)}\})\mathbf{x}^{(b)} \tag{4}$$

as long as the $b_{i,k}^{(s)}$ are chosen to also satisfy $x_k^{(s)}(b_{i,k}^{(s)}) \leq x_k^{(s)} \leq x_k^{(s)}(b_{i,k}^{(s)} + 1)$. These constraints will be used in section 6.1.6 which formulates the optimization problem.

We also allow a supplier to express additional linear constraints so that, for example, he may represent that he does not deliver on Sunday. Thus the supplier may define the matrices $\mathbf{G}_1^{(s)}(\boldsymbol{\varkappa})$, $\mathbf{G}_2^{(s)}(\boldsymbol{\varkappa})$, and the vectors $\mathbf{g}_1^{(s)}(\boldsymbol{\varkappa})$, $\mathbf{g}_2^{(s)}(\boldsymbol{\varkappa})$ such that $\mathbf{G}_1^{(s)}\mathbf{x}^{(s)} = \mathbf{g}_1^{(s)}$ and $\mathbf{G}_2^{(s)}\mathbf{x}^{(s)}\mathbf{x} \leq \mathbf{g}_2^{(s)}$. $\mathbf{G}_1^{(s)}(\boldsymbol{\varkappa})$ and $\mathbf{G}_2^{(s)}(\boldsymbol{\varkappa})$ have $c_1^{(s)}$ and $c_2^{(s)}$ rows respectively.

Discrete Capabilities

It is easy to imagine that a supplier's response on a discrete dimension is highly constrained by the values of the response on other dimensions, e.g., certain product characteristics come only in certain colors and package sizes. Consequently, it is not suitable to explicitly define a response but only to make available a supplier's constraints amongst the discrete variables. Consider 3 discrete dimensions where $\varkappa_1 \in \mathcal{D}_1 = [a, b, c]$, $\varkappa_2 \in \mathcal{D}_2 = [A, B, C, D]$, and $\varkappa_3 \in$

$\mathcal{D}_3 = [\alpha, \beta, \gamma, \delta]$, and assume the supplier has the following 3 constraints

$$\mathcal{C}_1(\varkappa_1, \varkappa_3) = \{(a, \alpha), (a, \beta), (b, \beta), (c, \beta)\},$$
$$\mathcal{C}_2(\varkappa_2, \varkappa_3) = \{(A, \beta), (B, \gamma), (D, \beta)\},$$
$$\mathcal{C}_3(\varkappa_1) = \{b, c\}.$$

We first note that there are 4 feasible solutions (or product configurations the supplier can meet): $[\varkappa_1, \varkappa_2, \varkappa_3] = [b, A, \beta]$, $[b, D, \beta]$, $[c, A, \beta]$, or $[c, D, \beta]$. Feasible solutions to the constraints define the response $\boldsymbol{\varkappa}^{(s)}$ for the discrete variables.

We indicate a supplier's or buyer's collective set of discrete constraints by $\mathcal{C}^{(s)}(\boldsymbol{\varkappa})$ and $\mathcal{C}^{(b)}(\boldsymbol{\varkappa})$ respectively.

6.1.6 The Optimization Problem

Having defined the necessary components, we now define the optimization task which determines the continuous $\mathbf{x}^*$ and discrete $\boldsymbol{\varkappa}^*$ parameters of the trade.

Since the trade must be acceptable to the supplier, we maximize the buyer's utility over a supplier's capabilities. Equivalently, we minimize the distance from the buyer's ideal values as

$$[\mathbf{x}^*, \boldsymbol{\varkappa}^*] = \arg\min_{\mathbf{x}^{(s)}, \boldsymbol{\varkappa}^{(s)}} \left\{ \left(\mathbf{x}^{(s)} - \boldsymbol{\mu}(\boldsymbol{\varkappa}^{(s)})\right)^t \mathbf{C}_w^{-1}(\boldsymbol{\varkappa}^{(s)}) \left(\mathbf{x}^{(s)} - \boldsymbol{\mu}(\boldsymbol{\varkappa}^{(s)})\right) + Q_d Z_w(\boldsymbol{\varkappa}^{(s)}) + Q_r R_w(\mathbf{w}) \right\}$$

where

$$\mathbf{x}^{(s)} = \left(\mathbf{I} - \mathbf{M}^{(s)}(\boldsymbol{\varkappa}^{(s)})\right)^{-1} \mathbf{c}(\boldsymbol{\varkappa}^{(s)}) + \left(\mathbf{I} - \mathbf{M}^{(s)}(\boldsymbol{\varkappa}^{(s)})\right)^{-1} \mathbf{M}^{(b)} \mathbf{x}^{(b)}$$

subject to the constraints over continuous variables

$$\mathbf{G}_1(\boldsymbol{\varkappa}^{(s)})\mathbf{x}^{(s)} = \mathbf{g}_1(\boldsymbol{\varkappa}^{(s)}), \qquad \mathbf{G}_2(\boldsymbol{\varkappa}^{(s)})\mathbf{x}^{(s)} \leq \mathbf{g}_2(\boldsymbol{\varkappa}^{(s)})$$

and the constraints over the discrete variables $\mathcal{C}^{(b)}(\mathbf{v}^{(s)})$, $\mathcal{C}^{(s)}(\mathbf{v}^{(s)})$. In the above, we have defined the $(c_1^{(s)} + c_1^{(b)}) \times n_c$ and $(c_2^{(s)} + c_2^{(b)}) \times n_c$ matrices $\mathbf{G}_1(\boldsymbol{\varkappa}^{(s)})$ and $\mathbf{G}_2(\boldsymbol{\varkappa}^{(s)})$ by

$$\mathbf{G}_1(\boldsymbol{\varkappa}^{(s)}) = \begin{bmatrix} \mathbf{G}_1^{(s)}(\boldsymbol{\varkappa}^{(s)}) \\ \mathbf{G}_1^{(b)}(\boldsymbol{\varkappa}^{(s)}) \end{bmatrix}, \quad \text{and} \quad \mathbf{G}_2(\boldsymbol{\varkappa}^{(s)}) = \begin{bmatrix} \mathbf{G}_2^{(s)}(\boldsymbol{\varkappa}^{(s)}) \\ \mathbf{G}_2^{(b)}(\boldsymbol{\varkappa}^{(s)}) \end{bmatrix}.$$

The $(c_1^{(s)} + c_1^{(b)})$- and $(c_1^{(s)} + c_1^{(b)})$-vectors $\mathbf{g}_1(\boldsymbol{\varkappa}^{(s)})$ and $\mathbf{g}_2(\boldsymbol{\varkappa}^{(s)})$ are defined by

$$\mathbf{g}_1(\boldsymbol{\varkappa}^{(s)}) = \begin{bmatrix} \mathbf{g}_1^{(s)}(\boldsymbol{\varkappa}^{(s)}) \\ \mathbf{g}_1^{(b)}(\boldsymbol{\varkappa}^{(s)}) \end{bmatrix}, \quad \text{and} \quad \mathbf{g}_2(\boldsymbol{\varkappa}^{(s)}) = \begin{bmatrix} \mathbf{g}_2^{(s)}(\boldsymbol{\varkappa}^{(s)}) \\ \mathbf{g}_2^{(b)}(\boldsymbol{\varkappa}^{(s)}) \end{bmatrix}.$$

The optimization is accomplished by iterating two distinct phases. Phase one sets the continuous parameters optimally for a given setting of the discrete variables. We define the

functions

$$d_1(\mathbf{x}, \boldsymbol{\varkappa}) = \big(\mathbf{x} - \boldsymbol{\mu}(\boldsymbol{\varkappa})\big)^t \mathbf{C}_w^{-1}(\boldsymbol{\varkappa})\big(\mathbf{x} - \boldsymbol{\mu}(\boldsymbol{\varkappa})\big) + R(\mathbf{r}; \boldsymbol{\varkappa}) \quad \text{and} \quad d_2(\boldsymbol{\varkappa}) = Q_d Z_w(\boldsymbol{\varkappa}^{(s)}),$$

The first phase of the optimization is the continuous problem:[12]

$$\mathbf{x}^*(\boldsymbol{\varkappa}) = \arg\min_{\mathbf{x}^{(s)}} d_1(\mathbf{x}^{(s)}, \boldsymbol{\varkappa}) \quad \text{subject to} \quad \mathbf{G}_1(\boldsymbol{\varkappa})\mathbf{x} = \mathbf{g}_1(\boldsymbol{\varkappa}) \text{ and } \mathbf{G}_2(\boldsymbol{\varkappa})\mathbf{x} \leq \mathbf{g}_2(\boldsymbol{\varkappa}). \quad (5)$$

A detailed discussion on the solution of the phase 1 optimization problem is found in appendix D. The second phase determines the best value of the discrete variables by minimizing over a function of $\boldsymbol{\varkappa}$ alone

$$\boldsymbol{\varkappa}^* = \arg\min_{\boldsymbol{\varkappa}} d_1(\mathbf{x}(\boldsymbol{\varkappa}), \boldsymbol{\varkappa}) + d_2(\boldsymbol{\varkappa}) \quad \text{subject to} \quad C^{(b)}(\boldsymbol{\varkappa}) \wedge C^{(s)}(\boldsymbol{\varkappa}). \quad (6)$$

Further details on the phase 2 optimization are given in Appendix E. Once $\boldsymbol{\varkappa}^*$ has been determined, we find $\mathbf{x}^*$ as $\mathbf{x}^* = \mathbf{x}(\boldsymbol{\varkappa}^*)$.

6.1.7 Aggregation

Often a buyer may be willing to divide an order between multiple suppliers in order to aggregate the required demand or to obtain better deals. In this section, we detail how the present invention supports this aggregate optimization.

Aggregation can only occur over the continuous variables where values may be subdivided. Each continuous variable x_i must be parcelled out amongst a set of suppliers. Consequently, we extend our notation to $x_i \rightarrow \tilde{x}_{i,k}$ giving the contribution of the kth supplier to continuous dimension i. The kth supplier may come from a (perhaps proper) subset of all suppliers. We indicate the set of potentially contributing suppliers as $\mathcal{K}$ and the number of potentially contributing suppliers as $|\mathcal{K}|$.[13]

We restrict the discrete variables to be the same across all potentially aggregated suppliers, i.e., we do *not* generalize $\varkappa_i \rightarrow \varkappa_{i,k}$. This simplifying assumption is made for two reasons. First, the size of the discrete optimization problem is smaller and so optimization be performed faster. Second, it may be difficult to elicit from the buyer the allowed discrete alternatives for each supplier. Nevertheless, this generalization is straightforward should the need arise.

This simplifying assumption requires that the union of discrete supplier constraints $C_{\mathcal{K}}(\boldsymbol{\varkappa}) \equiv \bigwedge_{k \in \mathcal{K}} C_k^{(s)}(\boldsymbol{\varkappa})$ yields a feasible solution when combined with the buyer's discrete constraints $C^{(b)}(\boldsymbol{\varkappa})$. A necessary (but not sufficient) condition for satisfaction is then that each constraint satisfaction problem k having constraints $C^{(b)}(\boldsymbol{\varkappa}) \wedge C_k^{(s)}(\boldsymbol{\varkappa})$ has a feasible solution.[14] Henceforth, we will assume that the set of suppliers, $\mathcal{K}$, satisfies this condition. If not, those suppliers violating the constraints $C^{(b)}(\boldsymbol{\varkappa}) \wedge C_k^{(s)}(\boldsymbol{\varkappa})$ are eliminated from consideration in $\mathcal{K}$.

[12] No optimization is required over range variables, since these are specified up front by both buyer and seller and merely add to the total distance.

[13] All work to this point is thus seen as the special case $|\mathcal{K}| = 1$.

[14] It is easily seen this requirement is not sufficient by having $C^{(b)} = \{\langle \varkappa_1, 1\rangle, \langle \varkappa_1, 2\rangle\}$, $C_1^{(s)} = \{\langle \varkappa_1, 1\rangle\}$, and $C_2^{(s)} = \{\langle \varkappa_1, 2\rangle\}$.

Discrete Search

We must search over the subsets of $\mathcal{K}$ for feasible solutions, which is a combinatorial problem. Fortunately, given a complete labelling of variables, determining the largest subset is easy. For any given labelling of all discrete variables, if each $\mathcal{C}^{(b)} \wedge \mathcal{C}_k^{(s)}\ \forall\ k \in \kappa \subset \mathcal{K}$ is satisfiable, then the union $\mathcal{C}^{(b)} \wedge \mathcal{C}_\kappa^{(s)}$ where $\mathcal{C}_\kappa^{(s)} = \bigwedge_{k\in\kappa} \mathcal{C}_k^{(s)}$ is also satisfiable under the same labelling. The largest subset of variables is found by adding all k which have feasible solutions with the buyer. We needn't worry about smaller subsets because the continuous optimization will assign zero values to those if appropriate. Consequently, for any given labelling $\varkappa$ we let $\mathcal{K}(\varkappa)$ represent the maximal subset of suppliers for which $\mathcal{C}^{(b)}(\varkappa) \wedge \mathcal{C}_\mathcal{K}(\varkappa)$ is satisfiable. It is this set of suppliers which enter into the continuous optimization. The number of participating suppliers is denoted by $|\mathcal{K}(\varkappa)|$.

Continuous Optimization

Optimization over the continuous variables is carried for each labelling $\varkappa$. Generally speaking, the buyer's utility will not be an explicit function of $x_{i;k}$ but only of x_i. We assume a linear relationship between these two quantities so that[15]

$$\mathbf{x} = \Xi\tilde{\mathbf{x}}.$$

The $n_c|\mathcal{K}(\varkappa)|$ vector $\tilde{x}$ is defined as $\tilde{\mathbf{x}}^t = [\tilde{\mathbf{x}}_1, \cdots, \tilde{\mathbf{x}}_{n_c}]$ where $\tilde{\mathbf{x}}_i^t = [\tilde{x}_{i;1}, \cdots, \tilde{x}_{i;|\mathcal{K}(\varkappa)|}]$. The $n_c \times n_c|\mathcal{K}(\varkappa)|$ matrix Ξ $\xi_{i,k}$ is assumed known and typically has the form[16]

$$\Xi = \begin{bmatrix} \boldsymbol{\xi}_1^t & \mathbf{0}^t & \cdots & \mathbf{0}^t \\ \mathbf{0}^t & \boldsymbol{\xi}_2^t & \cdots & \mathbf{0}^t \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}^t & \mathbf{0}^t & \cdots & \boldsymbol{\xi}_{n_c}^t \end{bmatrix}$$

where $\mathbf{0}$ is the K-vector of all zeros and $\boldsymbol{\xi}_i$ is the linear combination relating x_i to the $\tilde{x}_{i;k}$. Under our assumptions for Ξ, $x_i = \boldsymbol{\xi}_i^t\tilde{\mathbf{x}}_i$. In cases where the buyer wants to accumulate the results from suppliers (e.g., aggregating quantities) $\boldsymbol{\xi} = \mathbf{1}$ is the $|\mathcal{K}(\varkappa)|$-vector of all 1s. In other cases the buyer may take $\boldsymbol{\xi} = \mathbf{1}/|\mathcal{K}(\varkappa)|$ so that the time of delivery becomes the average time of delivery across the suppliers. Constraints on $\mathbf{x}$ become constraints on $\tilde{\mathbf{x}}$ via

$$\tilde{\mathbf{G}}_1(\varkappa)\tilde{\mathbf{x}} = \mathbf{g}_1(\varkappa) \quad \text{and} \quad \tilde{\mathbf{G}}_2(\varkappa)\tilde{\mathbf{x}} \leq \mathbf{g}_2(\varkappa) \tag{7}$$

where $\tilde{\mathbf{G}}_1(\varkappa) = \tilde{\mathbf{G}}_1(\varkappa)\Xi$ and $\tilde{\mathbf{G}}_1(\varkappa) = \tilde{\mathbf{G}}_1(\varkappa)\Xi$. We might also expect the buyer to add additional linear constraints, such as requiring the latest shipment from any supplier to arrive earlier than a certain date, or requiring all deliveries to arrive the same day. There can also be constraints specific to particular suppliers, e.g., the buyer doesn't want any more than 100 units from supplier 5. These can be handled simply as constraints on the individual $\tilde{x}_{i;k}$ and added as extra rows to $\tilde{\mathbf{G}}_1(\varkappa)$, $\tilde{\mathbf{G}}_2(\varkappa)$, $\tilde{\mathbf{g}}_1(\varkappa)$, and $\tilde{\mathbf{g}}_2(\varkappa)$.

[15] We can also relate $\mathbf{x}$ and $\tilde{\mathbf{x}}$ by $\mathbf{x} = \Xi\tilde{\mathbf{x}}+\boldsymbol{\nu}$ for some constant vector $\boldsymbol{\nu}$, which will not cause any complications. However, there seems to be little practical reason to do so.

[16] With no additional computational complexity, we can allow Ξ to depend on $\varkappa$.

With aggregation, the quadratic form to be minimized is $(\Xi\tilde{\mathbf{x}} - \boldsymbol{\mu}(\varkappa))^t \mathbf{C}_w^{-1}(\varkappa)(\Xi\tilde{\mathbf{x}} - \boldsymbol{\mu}(\varkappa))$ subject to the constraints given in Eq. (7). This minimization can be carried out through a straightforward generalization of the method given in Appendix D.

6.2 Implementation

In this section we outline an implementation of the entire e-procurement invention. We begin with a high-level description of the architecture, then fill in the details by describing a complete object model.

6.2.1 High-level Architecture of the Invention

There are at least two modes in which the invention may be used. First, the invention may reside at the site of large buyers, and suppliers who wish to sell to the buyer may be required to submit their capabilities via a web interface to the buyer. The invention may also be used within a marketplace hosted by a third party. Buyers/sellers log onto the market, submit their preference/capabilities, and act on the results. The architecture is modular enough to support both modes of operation.

In Figure 1 we present an architecture for the invention. We describe the architecture, starting from the optimization algorithm which finds matches between buyers and sellers and work our way outwards.

A controller surrounds the optimization engine, feeding it buyer preferences and seller capabilities. If multiple optimization processes are running (perhaps on different machines), the controller can also do load balancing, forwarding the request to the least busy process. The controller decomposes preferences and capabilities into their constituent buyer- and seller-specific versions (see below), selects the most specific matching preference/capability pairs, and sends them to the matching engine for optimization. The controller then collects responses from the matching engine and returns them to the buyer. Additionally, the controller logs all results into a database for recording purposes.

Another layer, called the Connector in Figure 1, separates the graphical user interface (GUI) through which users communicate with the tool from the controller. This layer serves a number of functions. The connector transforms the description of preferences and capabilities from the GUI into a form suitable for the implementation of the matching engine. Part of this transformation involves validation of appropriate input from the GUI layer so that no malformed input is ever sent to the controller. The Connector layer can also pull data from ERP or SCM systems and automatically infer preferences (using the total cost of ownership function) for the buyer. The enterprise abstraction layer insulates the Connector from the precise details of the manner in which the ERP and SCM data needs to be gathered. Total cost of ownership is evaluated in the simulation modules, which may either be running locally at the client's site or running centrally at the main server. These simulation modules pull operational data (the vector $\boldsymbol{\beta}$)from the enterprise abstraction layer. A preference optimization module (TCO) minimizes the total cost of ownership to determine the ideal trade and the flexibilities around the ideal trade.

At the outmost level, a layer provides integration with the GUI and/or host system. A number of administrative systems are expected at this layer. Market administration services allow easy definition of trading spaces, the dimensions of negotiation, limits on continuous variables, allowed settings of the discrete variables, etc. User administration services allow an administrator to define buyers, passwords, spending limits, etc. Supplier services accomplish similar tasks on the supply side. Managers for preferences, capabilities, and match results ensure that these objects are properly stored in a database. This layer layer also dynamically generates the html necessary for presentation of the data via a web interface to buyers and sellers.

For maximal portability, communications between the View and Connector are via XML documents. For maximal efficiency, communications between the Connector and matching controller are as serialized Java objects.

6.2.2 An Object Model for the Invention

The fundamental objects required for the invention are preferences from buyers, capabilities from sellers, and match results returned to all parties. The components of such objects have already been considered from a mathematical point of view, and we now describe one possible computer representation.

In this section we describe a complete grammar for the object model. The following syntactic conventions are used:

- ⟨nt⟩ denotes a non-terminal symbol nt
- [obj] denotes an optional grammar segment obj
- {obj} denotes 1, or many times the grammar segment obj
- $\rightarrow$ denotes a production rule for non-terminal symbol. If there are multiple rules, say ⟨a⟩, ⟨b⟩, and ⟨c⟩, then these are denoted as

$$\langle \text{nt} \rangle \rightarrow \langle a \rangle | \langle b \rangle | \langle c \rangle .$$

 In contrast, a production rule of the form

$$\langle \text{nt} \rangle \rightarrow \langle a \rangle, \langle b \rangle, \langle c \rangle$$

 indicates that the non-terminal ⟨nt⟩ is composed of three grammar segments, ⟨a⟩, ⟨b⟩, and ⟨c⟩

- terminal keywords are in serif font

Obvious non-terminal grammar elements like ⟨string⟩ and ⟨integer⟩ are not described.

Supply Side

To represent capabilities that apply to a specific buyer (perhaps for contractual reasons), we have defined a capability to be a list of ⟨buyerSpecificCapability⟩. With one exception, a

buyer-specific capability applies only to one buyer – that buyer associated in the id field of the ⟨buyerSpecificCapability⟩. The exception occurs if the id field is * or wildcard. This indicates that the capability applies to all buyers. Using buyer-specific capabilities, suppliers can represent specific capabilities to certain buyers and generic capabilities applying to all other buyers. By not including a wildcard ⟨buyerSpecificCapability⟩ and only listing ⟨buyerSpecificCapability⟩s applicable to specific buyers, sellers can also represent the fact that they will trade only with a subset of all buyers. In cases where both the wildcard ⟨buyerSpecificCapability⟩ and a ⟨buyerSpecificCapability⟩ applicable to a specific buyer apply, the most specific ⟨buyerSpecificCapability⟩ is selected.

A schematic of a ⟨sellerSpecificPreference⟩ is given in Figure 2.

We begin at the top level of a capability:

$$\text{capability} \rightarrow \{\langle\text{buyerSpecificCapability}\rangle\}$$

where

$$\begin{aligned}\langle\text{buyerSpecificCapability}\rangle \rightarrow\ & \text{id: } \langle\text{id}\rangle,\\ & \text{discrete: } \{\langle\text{discreteVarDescription}\rangle\},\\ & \text{continuous: } \{\langle\text{continuousVarDescription}\rangle\},\\ & \text{range: } \{\langle\text{rangeVarDescription}\rangle\},\\ & [\text{discreteConstraint: } \langle\text{discreteConstraint}\rangle],\\ & \text{instance: } \{\langle\text{discreteCapabilityInstance}\rangle\}\\ & [\text{aggregationParticipation: } 0|1].\end{aligned}$$

⟨id⟩ identifies a buyer or group of buyers. Individual buyers are represented by some unique identifier (say an integer) and the group of all buyers is identified by the wildcard character '*'. So we have

$$\langle\text{id}\rangle \rightarrow \langle\text{integer}\rangle \mid *\,.$$

aggregationParticipation is a Boolean flag giving the supplier's willingness to participate in aggregate orders to the identified buyer.

Each of the variable constituent components is described by

$$\begin{aligned}\langle\text{discreteVarDescription}\rangle \rightarrow\ & \text{name: } \langle\text{integer}\rangle,\\ & \text{allowedValues: } \{\langle\text{integer}\rangle\}\\ \langle\text{continuousVarDescription}\rangle \rightarrow\ & \text{name: } \langle\text{integer}\rangle,\\ & \text{min: } \langle\text{double}\rangle,\\ & \text{max: } \langle\text{double}\rangle\\ \langle\text{rangeVarDescription}\rangle \rightarrow\ & \text{name: } \langle\text{integer}\rangle..\end{aligned}$$

In its simplest form, a ⟨discreteConstraint⟩ is a list of more primitive constraints

$$\langle \text{discreteConstraint} \rangle \rightarrow \{\langle \text{primitiveDiscreteConstraint} \rangle\}$$

where each primitive constraint is composed as follows:

$$\begin{aligned} \langle \text{primitiveDiscreteConstraint} \rangle \rightarrow\ & \text{name: } \langle \text{string} \rangle \\ & \text{variables: } \{\langle \text{discreteVarName} \rangle\}, \\ & \text{includes: } 0 \mid 1, \\ & \text{values: } \langle \text{integerMatrix} \rangle \end{aligned}$$

⟨discreteVarName⟩ is the name of the discrete variable involved in the constraint

$$\langle \text{discreteVarName} \rangle \rightarrow \langle \text{integer} \rangle.$$

The includes field is a bit. If the bit is 1, then the combinations listed in the values field are the allowed values the variables may take on. If the bit is 0, then the combinations listed in values are the excluded combinations, i.e., everything in the powerset of the variables is allowed except those combinations listed in values. The order of the variable names is significant, since they will be assumed to be in the same order in values. If there are a variables involved in the constraint, and c constraints, then ⟨integerMatrix⟩ is an $a \times c$ matrix of integers:

$$\begin{aligned} \langle \text{integerMatrix} \rangle &\rightarrow \langle \text{integerVector} \rangle, \cdots, \langle \text{integerVector} \rangle \\ \langle \text{integerVector} \rangle &\rightarrow \langle \text{integer} \rangle, \cdots, \langle \text{integer} \rangle \end{aligned}$$

The ⟨discreteCapabilityInstance⟩ component is described by

$$\begin{aligned} \langle \text{discreteCapabilityInstance} \rangle \rightarrow\ & \text{mask: } \langle \text{discreteVarMask} \rangle, \\ & [\text{rangeCapability: } \{\langle \text{rangeVarInstance} \rangle\}], \\ & \text{continuousCapability: } \langle \text{continuousCapability} \rangle \\ & \text{continuousConstraints: } \langle \text{continuousConstraints} \rangle \end{aligned}$$

A ⟨rangeVarInstance⟩ defines a range variable and has the form

$$\begin{aligned} \langle \text{rangeVarInstance} \rangle \rightarrow\ & \text{name: } \langle \text{integer} \rangle, \\ & \text{min: } \langle \text{double} \rangle, \\ & \text{max: } \langle \text{double} \rangle. \end{aligned}$$

The ⟨discreteVarMask⟩ relates to the discussion of 6.2.2. As in Table 3 we have

$$\langle \text{discreteVarMask} \rangle \rightarrow \{\langle \text{extendedVarValue} \rangle\}$$

where an ⟨extendedVarValue⟩ is either an integer from the domain of the discrete variable or the wildcard character '*':

$$\langle \text{extendedVarValue} \rangle \rightarrow \langle \text{integer} \rangle \mid * .$$

⟨continuousConstraints⟩ describes the hard linear constraints for the continuous variables. Since these constraints may be either inequality or equality, we have

$$\langle \text{continuousConstraints} \rangle \rightarrow [\textsf{equality: } \langle \text{linearConstraints} \rangle], \\ [\textsf{inequality: } \langle \text{linearConstraints} \rangle]$$

Both the equality and inequality constraints are expressed through a matrix which is $c \times n_c$ where c is the number of constraints, and a vector which is $c \times 1$. Consequently we have

$$\langle \text{linearConstraints} \rangle \rightarrow \textsf{matrix: } \langle \text{doubleMatrix} \rangle, \\ \textsf{vector: } \langle \text{doubleVector} \rangle$$

A ⟨doubleMatrix⟩ is defined by

$$\langle \text{doubleMatrix} \rangle \rightarrow \langle \text{doubleVector} \rangle, \cdots, \langle \text{doubleVector} \rangle$$

and a ⟨doubleVector⟩ is just what the name suggests – a vector of doubles:

$$\langle \text{doubleVector} \rangle \rightarrow \langle \text{double} \rangle, \cdots, \langle \text{double} \rangle.$$

The only remaining undescribed element above is ⟨continuousCapability⟩ whose description is

$$\langle \text{continuousCapability} \rangle \rightarrow \textsf{breakPoints: } \langle \text{doubleListMatrix} \rangle, \\ \textsf{valAtBreakPoints: } \langle \text{doubleListMatrix} \rangle$$

⟨doubleListMatrix⟩ describes a $n_c \times n_c$ matrix whose elements are lists of ⟨double⟩:

$$\langle \text{doubleListMatrix} \rangle \rightarrow \langle \text{doubleListVector} \rangle, \cdots, \langle \text{doubleListVector} \rangle$$
$$\langle \text{doubleListVector} \rangle \rightarrow \langle \text{doubleList} \rangle, \cdots, \langle \text{doubleList} \rangle$$
$$\langle \text{doubleList} \rangle \rightarrow \{\langle \text{double} \rangle\}$$

It is assumed that the rows and columns of the matrix are in some canonical order so that we know which continuous variable is referenced. A natural order is the one defined in {⟨continuousVarDescription⟩}

Preferences

Just as capabilities may be buyer-specific so too may preferences be seller-specific. The same rules determining which seller-specific preference to apply are followed. A schematic of a

⟨sellerSpecificPreference⟩ is given in Figure 3.

We define a preference as follows

$$\langle \text{preference} \rangle \rightarrow \{\langle \text{sellerSpecificPreference} \rangle\}[, \langle \text{aggregatedPreference} \rangle]$$

i.e., a preference is a list of ⟨sellerSpecificPreference⟩ with an optional aggregated preference. We first describe ⟨sellerSpecificPreference⟩ and then consider ⟨aggregatedPreference⟩.

The ⟨sellerSpecificPreference⟩ is composed as follows

$$\begin{aligned}\langle \text{sellerSpecificPreference} \rangle \rightarrow & \text{id: } \langle \text{id} \rangle, \\ & \text{discrete: } \{\langle \text{discreteVarDescription} \rangle\}, \\ & \text{continuous: } \{\langle \text{continuousVarDescription} \rangle\}, \\ & \text{range: } \{\langle \text{rangeVarDescription} \rangle\}, \\ & \text{dimensionWeights: } \langle \text{dimensionWeights} \rangle, \\ & \text{discreteTradeoff: } \langle \text{tradeoffTables} \rangle \\ & [\text{discreteConstraint: } \langle \text{discreteConstraint} \rangle], \\ & \text{instance: } \{\langle \text{discretePreferenceInstance} \rangle\}\end{aligned}$$

Of these elements, only ⟨dimensionWeights⟩, ⟨tradeoffTables⟩, and ⟨discretePreferenceInstance⟩ have yet to be defined. ⟨dimensionWeights⟩ gives the weights of all dimensions that indicate their importance. For convenience we break up the weights according to the three types of variables. Thus we have

$$\begin{aligned}\langle \text{dimensionWeights} \rangle \rightarrow & \text{range: } \langle \text{doubleVector} \rangle, \\ & \text{discrete: } \langle \text{doubleVector} \rangle, \\ & \text{continuous: } \langle \text{doubleVector} \rangle\end{aligned}$$

A ⟨doubleVector⟩ has been described previously. Each of the corresponding vectors is as long as the number of range, discrete, or continuous dimensions. ⟨tradeoffTables⟩ is an $n_{\text{discrete}} \times n_{\text{discrete}}$ matrix of ⟨tradeoffTable⟩:

$$\begin{aligned}\langle \text{tradeoffTables} \rangle &\rightarrow \langle \text{tradeoffTableMatrix} \rangle \\ \langle \text{tradeoffTableMatrix} \rangle &\rightarrow \langle \text{tradeoffTableVector} \rangle, \cdots, \langle \text{tradeoffTableVector} \rangle \\ \langle \text{tradeoffTableVector} \rangle &\rightarrow \langle \text{tradeoffTable} \rangle, \cdots, \langle \text{tradeoffTable} \rangle\end{aligned}$$

A ⟨tradeoffTable⟩ is simply a matrix of double values.

Finally, we turn to the last undefined component of a ⟨preference⟩. A ⟨discretePreferenceInstance⟩

is composed as follows:

⟨discretePreferenceInstance⟩ → mask: ⟨mask⟩,
[rangeIdeal: {⟨rangeVarInstance⟩}],
continuousIdeal: ⟨doubleVector⟩,
tradeoffMatrix: ⟨doubleMatrix⟩,
[continuousConstraints: ⟨continuousConstraints⟩]

The rangeIdeal and continuousIdeal fields give the desired range and continuous trade parameters. The tradeoffMatrix field gives the positive definite matrix expressing the tradeoffs amongst the continuous variables. ⟨continuousConstraints⟩ have been described previously in the sell-side specification.

To complete the specification of preferences, we conclude with the definition of ⟨aggregatedPreference⟩. Refer to the discussion of section 6.1.7 for details.

⟨aggregatedPreference⟩ → participants: {⟨aggSpecification⟩},
contributionType: ⟨contributionTypeVector⟩,
additionalConstraints: ⟨continuousConstraints⟩,
discrete: {⟨discreteVarDescription⟩},
continuous: {⟨continuousVarDescription⟩},
range: {⟨rangeVarDescription⟩},
dimensionWeights: ⟨dimensionWeights⟩,
discreteTradeoff: ⟨tradeoffTables⟩
[discreteConstraint: ⟨discreteConstraint⟩],
instance: {⟨discretePreferenceInstance⟩}

In the above definition, the previously defined elements maintain their meaning. The additionalConstraints field allows the buyer to express constraints around the aggregation, such as "all orders must arrive on the same day," etc. participants lists the suppliers who can participate in the aggregation and their characteristics. Note that if the wildcard supplier participates, the order can potentially be aggregated across all suppliers. ⟨aggSpecification⟩ describes information specific to a supplier participating in the aggregation. It is defined by

⟨aggSpecification⟩ → id: ⟨id⟩.

id identifies the participating supplier and constraints specific to that supplier defined in an accompanying ⟨sellerSpecificPreference⟩ will be used in the optimization. Additional information may be added as required. The contributionType field is used to define the $\boldsymbol{\xi}$ vectors used in aggregation. The ⟨contributionTypeVector⟩ consists of n_c elements indicating the type of

discrete mask	output	specificity
$[* \quad * \quad *]$	{continuous1, range1}	0
$[\varkappa_1 \quad * \quad *]$	{continuous2, range2}	1
$[\varkappa_1 \quad * \quad \varkappa_3]$	{continuous3, range3}	2

Table 3: Example of discrete masks for specifying continuous and range variables which are dependent on discrete variables. $\varkappa_1$ and $\varkappa_3$ signify specific values for the first and third discrete variables. The specificity of each mask is indicated in the third column.

aggregation for each continuous dimension:

$$\langle \text{contributionTypeVector} \rangle \rightarrow \langle \text{contributionType} \rangle, \cdots, \langle \text{contributionType} \rangle.$$

Possible contribution types include

$$\langle \text{contributionType} \rangle \rightarrow \text{sum, average, zero.}$$

sum sets $\boldsymbol{\xi} = \mathbf{1}$, average sets $\boldsymbol{\xi} = \mathbf{1}/|\mathcal{K}(\boldsymbol{\varkappa})|$, and zero sets $\boldsymbol{\xi} = \mathbf{0}$.

Masking

We have allowed constraints, ideal values, tradeoffs, and continuous capabilities to be dependent on discrete variables. In this section we describe an efficient manner in which to encode this dependence.

The data structure must represent continuous and range variables for all valid discrete inputs. An efficient way to do this is to use hierarchical definitions. At the top of the hierarchy are the definitions of the continuous and range variables for the discrete values $\boldsymbol{\varkappa}^t = [*, \cdots, *]$. These values apply to all $\mathbf{v}$ unless more specialized masks are defined. A more specialized mask of the continuous and range variables is specified by defining values for some of the components $\varkappa_i$. The more components that are defined, the more specialized the definition. The most specific mask is always used. An example definition for three discrete variables is given in Table 3. The response to the lookup $[\tilde{\varkappa}_1 \; \tilde{\varkappa}_2 \; \tilde{\varkappa}_3]$ is {continuous3, range3} if and only if $\varkappa_1 = \tilde{\varkappa}_1 \wedge \varkappa_3 = \tilde{\varkappa}_3$, {continuous2, range2} if and only if $\varkappa_1 = \tilde{\varkappa}_1 \wedge \varkappa_3 \neq \tilde{\varkappa}_3$, and {continuous1, range1} otherwise.

Match Results

Returned to the buyer is a list of matches with different suppliers, which can be ranked and viewed in many different ways in the GUI. A ⟨matchResultList⟩ is a list of matchResult:

$$\langle \text{matchResultList} \rangle \rightarrow \{\langle \text{matchResult} \rangle\}.$$

A match result may either be a ⟨singleSupplierMatchResult⟩ or an ⟨aggregatedMatchResult⟩:

$$\langle \text{matchResult} \rangle \rightarrow \langle \text{singleSupplierMatchResult} \rangle | \langle \text{aggregatedMatchResult} \rangle.$$

A ⟨singleSupplierMatchResult⟩ represents the best trade with a single supplier and is composed

of the following elements:

⟨singleSupplierMatchResult⟩ → supplierId: ⟨integer⟩,
utility: ⟨double⟩,
feasible: 0|1,
[costFactors: {⟨double⟩],
continuous: {⟨double⟩},
discrete: {⟨discreteVarDescription⟩},
range: ⟨rangeVarInstance⟩.

The supplierId indicates the supplier sourcing this trade and the utility field indicates the utility of the trade (which can be used to rank the trades). feasible is a bit indicating whether or not a feasible trade with this supplier was found. The continuous, discrete, and range fields list the respective trade parameters determined by the matching algorithm. The optional cost factors field lists the constituent costs contributing to the total cost of ownership C_o evaluated at the trade point returned in the ⟨singleSupplierMatchResult⟩.

An ⟨aggregatedMatchResult⟩ returns the optimal trade when the buyer has requested aggregation. It is composed of the following elements:

⟨aggregatedMatchResult⟩ → utility: ⟨double⟩,
feasible: 0|1,
[costFactors: {⟨double⟩],
supplierTradeParameters: {⟨supplierTradeParameters⟩}.

As before, the utility field gives the utility of the aggregate trade, and the feasibility flag indicates whether or not a feasible aggregate trade was found (there may not be if the constraints were too stringent). costFactors can also be returned in C_o is sufficiently general to handle aggregated trades. Finally, ⟨supplierTradeParameters⟩ lists the trade parameters for each supplier involved in the aggregation. It is defined as follows:

⟨supplierTradeParameters⟩ → supplierId ⟨*integer*⟩,
continuous: {⟨double⟩},
discrete: {⟨discreteVarDescription⟩},
range: ⟨rangeVarInstance⟩.

6.3 Summary

We have described an efficient computational procedure in which to encode buyer's trading preferences and hard constraints, supplier's delivery capabilities and constraints, and optimize to find those matches between one buyer and one or many sellers that maximize the buyer's

utility. By optimizing against both qualitative and quantitative factors, and exploiting the trading flexibilities possessed by both parties, the system determines better trades. The tool is particularly useful as companies move their direct material purchasing online. By optimizing across flexibilities, win-win trades are discovered for both trading parties.

The representation of trading preferences is designed to be expressive yet easily elicitable from a buyer, and computationally tractable. The representation of supplier capabilities was chosen to parallel the manner in which suppliers already think of their delivery capabilities and seamlessly includes volume discounts and incremental costs. These supplier capabilities may be part of an online catalog. The representation of the negotiation space is rich, supporting three types of variables.

We have outlined a manner in which preferences may be inferred automatically through models of the purchasing company. Such models incorporate many cost factors, taking the total cost of ownership into account. The system provides trades which minimize the total cost and represent significant new savings.

The invention can operate both at a buyer's site, where suppliers input their capabilities through an HTML interface to the world wide web or as an embedded part of an electronic market hosted by a particular web site. The invention may operate at regularly scheduled intervals or sporadically in lieu of current request for quotations (RFQ). The buyer may broadcast a RFQ event to suppliers, indicating a time within which suppliers must respond. At the close of the event, the buyer can use the present invention to assist in the analysis of the supplier responses.

Complex algorithms have been specified which should permit most matching optimization to occur in near real-time. The rapidity of optimization, combined with graphical what-if tools, allows for analysis and exploration of trades, which should significantly improve the quality of purchasing decisions.

6.4 An e-Commerce Infrastructure for Value Chains

In this section we describe in detail how the proposed infrastructure delivers on the promises made in the Summary of the Invention. We begin by describing major innovations in the present invention and how they are all used synergistically.

6.4.1 Major Innovations

The most broad invention combines at least four advances:

1. multidimensional automated markets (hereafter simply markets) which capture many aspects of value.
2. algorithms and interfaces which implicitly allow for consumers to express the preferences over the multiple dimensions of value
3. linked markets allowing for complex assembly of products

4. specification of the constraints (both logical and numerical) inherent between markets to allow for coordinated buys and sells between markets

In addition, inventions described in a patent application titled, "An Adaptive and Reliable System and Method for Operations Management", Application Number 09/345,441 filed July 1, 1999 (the contents of which are herein incorporated by reference)can also be used in conjunction with the present invention. These other inventions are:

5. the use of models and optimization algorithms to optimally determine the best bids to submit to the automated market

6. the use of subset relations `is-a`, `has-a` *etc.* to automatically construct the constraints between markets

6.4.2 Integrated Picture

The internet and e-commerce are changing the way consumers and businesses trade with each other. One interesting trend is the development of centralized economic hubs (*e.g.* eSteel, ChemDex) through which all e-commerce in a particular domain flows. This trend is expected to continue and become more prevalent. The invention described here is the tool that drives these economic hubs.

Before describing the components and innovations in detail we describe the overall architecture of the integrated system. For concreteness we focus on the invention as applied to the personal computer (PC) supply chain. We note that the ideas can be applied equally well to supply chains in other industries.

The infrastructure is composed of three major components all running on different computers. A central server (or more likely servers) coupled to the economic hub serves as the source of consumer demand. Other sets of servers act as the trading markets themselves relaying information between buyers and sellers. Finally, running at each market participant's site is client software coupled to the relevant markets.

Consumer Demand: Consumer demand is pulled through the supply chain through a set of coupled reverse auction [17]markets. An interface iteratively elicits multi-dimensional consumer preferences which is then forwarded to a top level market where suppliers offer to fulfill the consumer request. The functioning of the consumer interface is described in more detail in section 6.4.3.

Markets: Markets within the supply chain are represented by servers which link trading partners. The market broadcasts demand, collects responses, and forwards results back to the source of the demand. See section 6.4.4 for more information on the functioning of markets, particularly the coupling between markets.

Client Companies: Each trading participant (whether buying or selling) is represented by client software connected to the appropriate markets. The client software both initiates

[17]A reverse auction is like an auction except that a request is broadcast to suppliers who bid on fulfilling the request. Unlike traditional auctions the proposed market clears in one shot, *i.e.* each supplier has only one chance to bid.

requests and responds to requests from other market participants. The client software running at the company also maintains a memory of all past transactions and can eliminate those that are out of date. A user interface allows companies to define the operation of their interface to the markets. See 6.4.5 for further details.

Supplier Push As has been mentioned the infrastructure is organized around demand pulled through the supply chain by the consumer. Interestingly, the same pull infrastructure can also be used by suppliers to push demand. In addition to requesting and responding to requests, the market interface running locally at company sites could also be used to advertise capability (capability not in response to any particular demand) to the market which will then forward the advertised capability to any participant connected to the market.

6.4.3 Climbing in Preference Space: the consumer interface

The consumer drives the integrated infrastructure by generating demand. In our example, a graphical user interface (GUI) at the PC economic hub provides a centralized interface through which all computers are purchased. The GUI provides a number of advantages to both consumers and computer vendors. A computer is described across multiple dimensions of value. Some of these dimensions include: price, memory size, processor speed, hard drive side, removable storage, monitor size and resolution, financing, warranty, delivery date, *etc.* For each dimension the user can identify a preference. For example the consumer may wish 128Mb of memory, a 600MHz Pentium III processor with a 15Gb hard drive for $1500 delivered within 3 weeks. Additionally, it may be important for the consumer to express constraints that must be satisfied. To keep things as simple as possible we express an optional inequality constraint for each dimension, *e.g.* price must be less than $1600 and the hard drive must be at least 12Gb.

Optional Dimensions: This example brings up an interesting feature of the dimensions of value. Imagine for a moment that one of the dimensions was portability with possible values being high portability, medium portability, and low portability. In each of these cases we might consider lightweight laptops, heavy laptops, and desktops. Any one of these choices might then invoke additional situational dimensions. For example, if we select high portability then battery life (in hours) becomes an important dimension which is only applicable to laptops. There are a number of ways to treat these optional dimensions. The simplest possibility is to have separate markets for laptops and desktops with appropriate dimensions for each. A better solution is to have a GUI which only turns on the optional dimensions when appropriate. If low portability is selected by the user the battery life dimension remains lightly greyed out and inaccessible. If medium or high portability is required then the battery life dimension is activated. In the technical details section we show how this can easily be accomplished.

Dimension Weighting: Optionally, the user may also specify an importance to each dimension. If hard drive size and price are paramount these two dimensions have high value while all other dimensions have low value. The weighting is used to identify similarity between computers; two computers which are quite different only in one dimension which is not highly

weighted by the consumer are considered quite similar. See the technical details section.

Threshold Constraints: A final way in which consumers may specify their needs is through constraints expressed for each dimension. The constraints are simple to state. The consumer specifies a threshold and whether or not he wants to be above or below that threshold. For example the consumer may specify that the monitor size has to be at least 17 inches and he will not pay any more than $1500. Knowledgeable consumers may define all these settings directly or the software may infer settings for less experienced consumers based upon simple questions (like the major uses of the computer).

Branding: Consumers may also have brand preferences for a computer or any of its components. The proposed GUI allows for users to express these preferences. There are two alternative ways in which branding might work and depending on the application either may be appropriate. In the first alternative brand name is a hard constraint and no computers are shown to the consumer unless the brand name matches the consumers desire. In the second alternative brand name is a soft constraint and the consumer merely prefers one brand over another.

In either case the GUI allows the user to specify for each dimension whether or not brand matters and if so which brand/brands is/are preferred.[18] If brand is a hard constraint this constraint is propagated to the top level market and all suppliers of the top level market so that no responses not respecting the constraint are ever returned to the consumer. If brand is a soft constraint it is incorporated into the distance function d_i as described in the technical details section. Otherwise identical components which match the desired brand are closer than those components which do not match the desired brand.

Iterative Hillclimbing and Recombination: Once the consumer has specified a desired computer defined as a point (and perhaps weights and constraints) in the high dimensional value space, a query is sent to a top-level automated market. Sellers of computers (*e.g.* Micron, Gateway, Dell) attached to the market respond to the consumer request by returning offers (one or many) to the market for candidate computers again represented in the high-dimensional space of value. Responses to the consumer are filtered to satisfy the specified constraints. The GUI allows the consumer to sort responses based on any of the dimensions (see the d_i function in the technical details section) or according to a global measure of distance between two computers (see the d function in the technical details section).

The consumer may modify any of the responses (a mutation) and use this as a resubmission to the market. This method of exploring the space of possible computers according to the consumer's desires and the supplier's availability can broadly be summarized as climbing in preference space. The climbing can begin from the most recently visited computer configuration or from any configuration that has been saved in a history list.[19] It is also perfectly feasible to allow the consumer to recombine desirable computer configurations. For example the user may ask for configurations which are "in between" two previously examined configurations and combine the advantages of both configurations. We also might allow the consumer to

[18] We might also allow consumers to rank their brand preferences but this extension seems unnecessary in light of the iterative nature of product exploration.

[19] History lists extend both within and between sessions.

submit more than a single configuration to the top level market and receive responses around all submissions.

Benefits of the Approach: This procedure offers a significant benefit over other approaches to multidimensional markets. In systems like OptiMark[20] the consumer (in this case a financial trader) must a priori specify preferences over all dimensions (price and quantity). The explicit specification of preference over even two dimensions is a difficult task and it is unreasonable to expect consumers to be able to do this. The present innovation allows consumers to iteratively explore preference space thereby implicitly defining preference in a friendly manner.

This method of interactive consumer exploration is suitable for any complex product and we expect the application to be broad. The present invention has many other applications[21]

Technical Details We have mentioned that computer configurations can be sorted on all dimensions according to how closely they match the consumers preferences. In this section we describe how this sorting might work and how it interacts with the weights and brand preferences attached to each dimension.

The value space defines a notion of distance or similarity between products (in this case computers) defined as follows: If there are a total of D dimensions[22] of value then a computer $\mathbf{c}$ is described as a D-vector of the settings for each dimension. Let i label the components of $\mathbf{c}$ so that for example c_1 = memory is c_2 = processor speed *etc.* For example, the memory dimension might assume any of the values $c_1 = 64$Mb, $c_1 = 128$Mb, or $c_1 = 256$Mb and similarly for all other dimensions. Let a_i be a binary variable (*i.e.* a_i takes the value 0 or 1) indicating which dimensions are actually used. If c_j is the dimension corresponding to battery life in hours then $a_j = 1$ indicates this dimension is appropriate while $a_j = 0$ indicates the dimension is not appropriate. The vector whose components are the a_i is represented by $\mathbf{a}$.

A consumer preference also includes optional weights and preferred brand. The weight (or importance) of dimension i is indicated by w_i. All w_i are non-negative and normalized so that $\sum_{i=1}^{D} a_i w_i = 1$. The vector of all weights is represented by $\mathbf{w}$. The brand preferences for dimension i are expressed as a (possibly empty) set $\mathcal{B}_i$ which includes desired brands. $\boldsymbol{\mathcal{B}}$ is the set of all $\mathcal{B}_i$. A complete consumer description is thus represented by the vector $\boldsymbol{\mathcal{C}} = \{\mathbf{c}, \mathbf{a}, \mathbf{w}, \boldsymbol{\mathcal{B}}\}$ with components $\mathcal{C}_i = \{c_i, a_i, w_i, \mathcal{B}_i\}$.

The distance $d(\mathbf{c}, \mathbf{c}')$ between computers $\mathbf{c}$ and $\mathbf{c}'$ is then defined as

$$d(\boldsymbol{\mathcal{C}}, \boldsymbol{\mathcal{C}}') = \frac{\sum_{i=1}^{D} a_i w_i \, d_i(\mathcal{C}_i, \mathcal{C}_i')}{\sum_{i=1}^{D} a_i} \tag{8}$$

where $d_i(\mathcal{C}_i, \mathcal{C}_i')$ measures the distance along the single dimension i. To allow for standard comparison between computer configurations we normalize all d_i so that they lie between 0 and 1.

[20] U.S. Patent 5,845,266, "Crossing network utilizing satisfaction density profile with price discovery features" dated 12/1/98.

[21] *e.g.* credit cards where the dimensions might be interest rate, yearly fee, grace period, benefits (like frequent flyer miles, *etc.*).

[22] The D dimensions include both required and optional dimensions.

We recall from previous discussion that for soft brand constraints the matching of a brand to its desired value is incorporated into the distance function. Thus we generally write the distance function d_i as the sum of two terms, one for brand and one for similarity. If b_i is an indicator variable whose value is 1 if the consumer has identified that brand matters for the ith dimension (*i.e.* $\mathcal{B}_i$ is non-empty) and 0 otherwise, then we write

$$d_i(\mathcal{C}_i, \mathcal{C}'_i) = \alpha_i b_i d_i^{(b)}(\mathcal{B}_i, \mathcal{B}'_i) + (1 - \alpha_i b_i) d_i^{(s)}(c_i, c'_i). \tag{9}$$

The α_i parameter determines the relative importance between brand name and similarity. Lacking better information we might take all $\alpha_i = 1/2$.[23]

For simplicity we assume that the brand distance function, $d_i^{(b)}(c_i, c'_i)$, is 0 if there is overlap in the preferred brands for c_i and c'_i and 1 otherwise, *i.e.*

$$d_i^{(b)}(\mathcal{B}_i, \mathcal{B}'_i) = \begin{cases} 1 & \text{if } \mathcal{B}_i \cap \mathcal{B}'_i = \emptyset \\ 0 & \text{otherwise} \end{cases} . \tag{10}$$

The similarity distance measure, $d_i^{(s)}(c_i, c'_i)$, is only slightly more complicated. The allowed values for most dimensions are discrete over a finite range. If the ith dimension is bounded and ranges from $c_i^{\min}$ to $c_i^{\max}$ then we might define

$$d_i^{(s)}(c_i, c'_i) \equiv \frac{|c_i - c'_i|}{c_i^{\max} - c_i^{\min}} \tag{11}$$

though other choices are certainly possible and may be more appropriate depending on the nature of the dimension. In many situations it is appropriate that the distance function is not symmetric, *i.e.* $d_i^{(s)}(c_i, c'_i) \neq d_i^{(s)}(c'_i, c_i)$. Examples of asymmetry are easy to find. A consumer may be more than willing to accept an extra few Mb of memory but will be very dissatisfied receiving a few Mb less. As a concrete example we may offer open ended ranges on most choices. The range is defined by a single point x'_i and an indication of whether the range is above or below that point. In the case of the range $[x'_i, \infty)$ we can take the distance function measuring the distance from the range to be $d(x_i, x'_i) = (x'_i - x_i)\theta(x'_i - x_i)$. In the opposite case where the range is $[0, x'_i]$ the distance function can be $d(x_i, x'_i) = (x_i - x'_i)\theta(x_i - x'_i)$. Accordingly, the present invention also includes possibilities for $d_i^{(s)}(c_i, c'_i)$.

Distance Ranking: The distance metric of Eq. (8) allows for a global ranking (after configurations not satisfying the constraints have been filtered out) of vendor responses based upon their similarity to the original consumer request (smaller distances are more similar). If $\mathcal{C}_{\text{req}}$ describes the consumer specified computer and α labels responses from vendors in response to the request then the automated market sorts the responses based upon $d(\mathcal{C}_{\text{req}}, \mathcal{C}_\alpha)$ and returns a specified number back to the consumer interface. The interface also allows sorting (in order of decreasing distance) on each dimension according to the d_i. The consumer then iterates the process starting from any of the returned offers, modifying them slightly

[23] Better values for α_i might be identified from real consumer interaction with the GUI.

and re-submitting to the market. In this way the consumer can move about in the space of preferences climbing towards ever more desirable computer configurations. When the consumer finally does identify a configuration that he desires to purchase it is a simple matter of hitting a buy button.

6.4.4 Integrating the Supply Chain: cascading markets

It is natural to consider the extension of high-dimensional automated markets to other transactions made in the supply chain. In fact, this may be essential to fulfill the kind of real-time consumer preference exploration described in section 6.4.3. These other markets will also typically be high dimensional but the dimensions will differ from the top-level market. For example there may be a hard drive market where buyers of hard drives (whether computer vendors, consumers, or value added retailers) meet sellers (*e.g.* suppliers like Seagate, Western Digital). The dimensions of the hard drive market might include: price, size (in Gb), type (IDE vs SCSI), quantity, delivery date, *etc.*

Operation of Cascading Markets: The basic idea is straightforward. Before a computer vendor responds to consumer requests in the top-level market the vendor may send its own requests to submarkets to purchase products and services it needs to meet the consumer demand. Suppliers to the computer vendors may in turn go to other submarkets before responding to computer vendor requests. A single consumer request may then result in real-time cascading inquiries to a tree of nested markets all the way down to the beginning of the supply chain. We call the market which originated a request the parent market and the submarket a child market. It is a requirement that any possible path through the cascading markets is acyclic.

When the consumer moves on to examine a new configuration all relevant market participants (*i.e.* those that responded to the initial customer request or any concommittant subsequent request) are informed and their offers are released. If, on the other hand, the consumer elects to purchase the configuration all participants are informed and are held to their offers. When combined with emerging XML standards for e-commerce (*e.g.* RosettaNet in the IT industry) the necessary "paperwork" for confirmed purchases can be generated by the computer and sent between all relevant parties (including the consumer after a credit card check).

Note that it remains unlikely that every consumer purchase will trigger a cascade of transactions throughout the tree of markets because it is not profitable in most supply chains to order and transport in single unit batches. Thus, only when inventories need to be replenished will companies in the supply chain go to the automated market in response to a request.

Auxiliary Information: It is useful to pass additional information between markets beyond merely the parameters describing the trade. There are many reasons extra information may be desired and we describe two applications of auxiliary information. Most simply, in cases where it is important to identify who the buyer or seller actually are an additional identifier may be passed labeling each trader.

A more serious application addresses a potential problem of nested markets. A problem with cascading markets is that demand can spuriously be amplified as it moves between

markets and away from its originating source. Consider a consumer demand for 100 computers. If two of the computer suppliers are both low on hard drives they might both submit a request to the hard drive market for an additional 100 drives. If the hard drive company were to estimate demand based solely upon the hard drive market then in this case the company would falsely assume that the demand was twice as high as it really is. This problem is easily fixed by supplying auxiliary information along with trade parameters. A request that comes into a toplevel market is assigned an identifier indicating the market and a unique identifier.[24]Every request to submarkets that is in response to a request in the parent market is again assigned both market and trade identifiers. These identifiers are appended to the identifiers from the parent market when the request is broadcast to all suppliers listening to the market. Returning to the hard drive example, if the top-level market has market identifier ASSEMBLE and the hard drive market has identifier HARDDRIVE and if the original consumer request had trade identifier 4561133 and the market trade identifiers were 56891 and 56892 respectively then the identifiers which are passed to all hard drive suppliers in response to both hard drive demands are ASSEMBLE:4561133,HARDDRIVE:56891 and ASSEMBLE:4561133,HARDDRIVE:56892 respectively. In this way all hard drive suppliers can see that the request originated with the same demand. In fact such mechanisms might also allow for improved forecasting of demand since all demand is accompanied by its path through the entire supply chain which may contain additional useful information.

Expiration Dates: One of the advantages of the present e-commerce infrastructure is the real-time setting of trade parameters. Companies can use the system to optimize their operations continuously. To achieve the maximal benefit from continuous optimization it is important that the offers made by any market participant have a bounded lifetime. What is optimal now may not be in a day (or even an hour). Thus an additional application of auxiliary information is the inclusion of expiration dates after which the trader parameters are invalid. Unless perishable trades are executed before the expiration date they are removed from the system.

Other B2B Market Issues The cascading submarkets are all business to business markets and as such can operate under different rules than the business to consumer market. In this section we address what some of these differences might be.

Clearing Mechanisms: We have proposed that the B2B markets clear in a fashion similar to the B2C market. One important difference is that these markets probably must clear in one shot in order to deliver real-time feedback to the end consumer. There are certainly other ways in which the B2B markets clear. One possibility is to use optimization tools to define satisfaction surfaces just as in OptiMark. In this way the parameters of the trade are determined by software to optimize some joint criteria. Ideally, we would like protection for other B2B market clearing mechanisms.

Aggregation: Another potentially important function of automated markets is the aggregation of orders to meet demand. For large orders it may be the case that no single supplier can meet

[24]All trades in that market will have distinct identifiers.

the requested demand. Market software can automatically collect responses to fulfill orders. This software, in addition to coordinating between purchases in different markets (see section 6.4.5) can also be used to coordinate multiple orders within the same market.

Many-to-many markets: Thus far the market mechanisms we have described all result in one-to-one trades (except if we aggregate orders in which case a trade may be many-to-one). For future reference we also note that the markets themselves may be more complex and used for many-to-many trades.

6.4.5 Coordination Between Markets: inter-market language

The assembly of a product or service from constituent products or services often requires coordinated purchases from a number of supply markets.

Logical Constraints: As illustration, there is no point in ordering a part if transportation can not also be arranged at the appropriate time. Both events must occur or else neither of the purchases should be made. This is an example of a Boolean logical constraint (in this case the logical and function) that exists between multiple markets. Of course in more complicated situations there may be more complex logical constraints between markets.

For added expressivity we propose using linear logic as the language in which to express logical constraints. Linear logic offers compelling advantages over Boolean logic since it subsumes Boolean logic and is resource sensitive exactly as real supply chain transactions are. Efficient implementations of linear logic exist so that checking of linear logical constraints is straightforward. The present invention includes support for both Boolean and linear logical constraints.

Numerical Constraints: There is another important class of relationships that may exist between markets. In most complex processes certain events have to occur sequentially; the process can not progress until some condition is fulfilled. Our shipping example also provides an example of this. There is no point arranging the purchase of transportation if the only available pick up date is before the item to be shipped can be completed. This is an example of a numerical constraint that exists between the parameters of transactions on different markets and optionally some description of the state of the company. Other examples might include cost constraints which require that the total expenditures on all purchases to be less than some threshold.

A Constraint Language: Any automated market solution to supply chain coordination problems will require these constraint mechanisms and generically these constraints will be either logical (as in the and constraint) or numerical (as in the cost constraint). We propose an Inter-Market Language (IML) in which to express these constraints so that the nested market solution always respects the user specified constraints.

IML constraints are specified as follows. Let m_i be a logical variable which is true if a transaction is made on the ith market (and false otherwise) and let $m_i(\mathbf{x}_i)$ be a logical variable which is true if the transaction occurs with multidimensional parameters $\mathbf{x}_i$.[25] Furthermore let

[25] m_i and $m_i(\mathbf{x}_i)$ are related by $m_i = \vee_{\mathbf{x}_i} m_i(\mathbf{x})$ where by $\vee_{\mathbf{x}_i}$ we mean the logical or of $m_i(\mathbf{x}_i)$ over all possible values of the parameters $\mathbf{x}_i$.

s be a vector of numerical values (for example describing current inventory levels, lead times, *etc*) and $\tilde{s}$ be a vector of logical values describing the state of the company. The two classes of IML constraints which both must simultaneously be satisfied are then expressed as

$$B(m_1, \cdots, m_M, \tilde{s}) \tag{12}$$

$$N_{=}(\mathbf{x}_1, \cdots, \mathbf{x}_M, \mathbf{s}) = 0, \quad N_{<}(\mathbf{x}_1, \cdots, \mathbf{x}_M, \mathbf{s}) < 0 \tag{13}$$

where $B(\cdot)$ is an arbitrary logical function, there are M relevant submarkets, and the functions $N_{=}(\cdot) = 0$ and $N_{<}(\cdot) < 0$ express the equality and inequality constraints respectively.

Operational Use of Constraints: Operationally the coordination amongst different market transactions is achieved as follows. Software representing the interests of a vendor (and therefore probably running at the vendors site) expressed in IML listens to servers representing various markets. When a market request that the vendor wishes to respond to is heard the software examines the companies internal state $\{\mathbf{s}, \tilde{\mathbf{s}}\}$ and determines what items, if any, need to be purchased on child submarkets. The software defines the purchase by defining a point or the desired parameters for each trade in each market. These requests are sent to servers representing each child market. The client software running at each company then waits for responses from each child market. If there are M total markets queried and n_i responses from the ith market then there are a total of $\prod_{i=1}^{M} n_i$ possible combinations of deals. The software then examines each deal filtering out those combinations which fail to satisfy the constraints of Eqs. (12) and (13). Thus we refer to this aspect of the client software as filtering. Those combinations which pass the constraints can then be ranked (either by a human user or by additional company specific criteria (see reliability below and in section 6.4.7) added to the client software running at the vendor site) and sent back as responses to the original request.

We have operationally described how constraints and filters may be used to coordinate purchases from automated markets. The converse problem is the coordination of sell orders to automated markets. This is less likely to be problematic since usually there is only one sell order being responded to at any given time and thus there are no sell-side coordination problems. Nevertheless, if there are sell-side coordinating activities that must occur the same mechanism can be used. A separate set of logical and numerical constraints can be defined in IML for sell orders. Any responses to requests can then be checked against these constraints to ensure no responses are sent which violate the selling practices expressed through the IML constraints. Sell-side filters can be used to enforce preexisting contracts like specially arranged pricing for preferred customers. This may be particularly important when responses to requests are submitted automatically by software coupled to the companies ERP software. Typically such software will appropriately construct a response but when no human is present to check responses sell-side filters provide a final sanity check.

Other Applications of Constraints: An important consideration for any company in a supply chain is reliability. The company must be assured that its suppliers meet their obligations. In current supply chains this consideration is important enough that companies often limit themselves to a handful of trading partners with whom they maintain long term

relationships. In order to fully reap the benefits of automated markets where many vendors may fulfill a need we need to resolve the issue of reliability. Filters are one way to address this issues. If for some historical reason a particular company doesn't like some suppliers the company may add these to be filtered. Thus the internal state $\{\mathbf{s}, \tilde{\mathbf{s}}\}$ may include references to unwanted suppliers. Alternatively, reliability may be used to rank bids that pass all filters.

6.4.6 Practical Concerns

In this section we show how real world concerns of scalability, real-time performance, reliability, and security can be assured within the proposed e-commerce framework.

Scalability The utility of an automated market is directly dependent on the liquidity (volume of trades) of the market. The greater the number of participants the greater the value to any individual participant. It is thus essential that the system scale gracefully to large numbers of market participants. Because the proposed e-commerce infrastructure is distributed scaling to large sizes is not a problem.

The market is driven by consumer demand pulled through the supply chain. Interfaces to consumer interaction can run locally on the consumer's computer as Java language applets. The central server (or servers) for consumer interaction is then only responsible for relaying consumer information (expressed in compact XML documents) into the top level market and returning the list of consumer responses to the appropriate user computer. The workload is shared across multiple machines with more machines involved the greater the number of consumers.

Similarly, the automated markets themselves serve mainly as relays of information and have little computational demands placed upon them. The markets send out requests, collect responses, and inform winning bidders of actual purchases. The bulk of the necessary computation is in the software which determines the parameters of the bids and responses. Since this software runs locally on each company's computer the system naturally exploits the parallelism inherent with increased vendor participation.

Real-Time Response To be most useful to consumers it is important that consumer feedback is in real or near real time. This may be problematic on the rare occasions in which a query triggers a cascade down through submarkets. Before the top level market can clear (*i.e.* for responses to come back to the consumer) the tier one market must clear and before the tier one market can clear the tier two market must clear, *etc.* Fortunately, it is likely that all B2B markets will be monitored by software so that turnarounds on all markets will be fast. Market time-outs beyond which responses will not be accepted ensure timely responses from all submarkets.

Nevertheless, in situations where the response time is slow we can offer the consumer an immediate but approximate response and warn the consumer that the response may not be accurate. This can be accomplished by caching previous responses and estimating the response

by interpolation of similar previous responses. Quick algorithms for the interpolation are easy to devise as long as the cached data can be accessed quickly as in any modern database.

6.4.7 Reliability

For any company to rely on the proposed infrastructure to run its supply chain it must be assured that the system is always up and running. For the system to be functioning for a company its own local server has to be up as well and all markets must be running. Each company is responsible for ensuring that its local servers are operating. The markets themselves will be served by a network of computers sharing the responsibility for managing transactions. The software which manages transactions can ensure that if a connection is broken during mid-transaction (*e.g.* if a computer goes down) it will be correctly reconstituted and resent. Such software is currently available from software companies.

6.4.8 Security

Whenever real dollars are being exchanged electronically there will always be important issues around security. We address each of these issues.

Commitment Assurance: It is important the every participant is confident that contra parties honor their commitments whether in terms of payment or delivery. We have described one mechanism that allows companies to automatically filter out those companies with whom they have had bad past experiences. It might also be expected that word of any party defaulting on its commitments would rapidly spread to all other participants making any default very costly for the offender.[26] If these mechanisms are insufficient the infrastructure may require each participant to pay an up front fee from which offenders compensate their contra partners.

Authentification: The system must ensure that all parties are who they say they are. This is an old problem faced by internet participants and has been solved satisfactorily by systems like PGP *etc.*

Privacy: It is also important that companies private data remains private. If a competitor could intercept a companies responses to the market it would have a distinct and unfair advantage. All communications will be strongly encrypted to assure that this can not occur.

Information Visibility: Similarly we may want different companies to have differing access to information.[27] For example data about consumer demand may be obtainable by a participant but only at a fee. This information may be served based on the authentification of the participant.

6.4.9 Optimization of the Supply Chain

The final ingredient in the infrastructure has been alluded to but not explicitly discussed. To achieve maximal benefit from the flexibility the proposed invention introduces at all levels of

[26] We might post defaults publicly so that other parties rapidly become aware of this knowledge and add the offender to their filters.

[27] This will certainly be the case if we wish to charge for access to some of this information.

the supply chain, real-time optimization systems can be used. This innovation is described in a patent application, titled, "An Adaptive and Reliable System and Method for Operations Management", Application Number 09/345,441 filed July 1, 1999 (the contents of which are herein incorporated by reference).

Coupling to ERP Software: The optimization systems monitor (in real-time) the state of the company and extrapolate from models of the company in its supply chain the optimal responses to requests that come in on the automated markets that the company supplies to. To achieve the best benefit, the servers or filters running at each company are coupled to ERP software to determine the best real-time response. The optimal response will be a function of the current instantaneous state of the company (as recorded in the ERP software) coupled with modeling and planning capabilities. A model of the likely future short term evolution of the state of the company may be used to best exploit the flexibility inherent in meeting the demand from the automated market. Such models might also take into account future expected demand and future responses from suppliers.

Another important optimization that might yield significant payoffs is in the tradeoff between making a response that is very close to meeting a requester's demand versus the savings obtained by perhaps being further from the requesters true desires.

6.4.10 Self-Organizing Supply Chains

In the procedure outlined above it is the responsibility of each company to define its business rules as expressed though its operating constraints and ranking criteria. This is not a unreasonable request and is a formalization of the company's current operating procedures. However, the present system does suggest a novel extension for the supply chain of the future. We can view the activities along a supply chain resulting in a finished good or service as an exercise in theorem proving in linear logic. Each company in the supply chain expresses its capabilities as predicates and functions (to include constraint) in linear logic. The finished good or service is a logical expression of a set of predicates. A path or method of assembly of the final good can then be represented as a proof of the logical expression representing the finished good or service. If the good can not be built then no such proof will be found. The precise manner in which the product is assembled is read off the proof net describing all the necessary steps. The beauty of this approach is its simplicity and the availability of theorem provers for linear logic. Though theorem proving for linear logic can be NP complete proofs of supply chains may be much simpler. To summarize: the mechanism of self-organization in the supply chain is *proving* the final good or service.

What are some of the axioms describing operations in the supply chain? There is shipping, manufacturing from components, etc. The state of the company includes inventory levels, lead times, etc.

6.4.11 Learning in the Infrastructure

There are many ways in which the data naturally recorded by the proposed infrastructure can be usefully used.

Prediction of customer demand: All the markets (both B2B and B2C) are rich sources of data for mining. Minimally, within any market we can use the data for both forecasting and inference of true customer desires (i.e. what they really want and now just what they are buying). The quantitative and nature of this market data should make mining it simpler than traditional data sources.

Inference of consumer preferences: The present invention also improves upon the efficiency of the end consumers search by aiding the search with learning tools. Based on the consumers interaction with the top-level market software can extrapolate to guess preferences and speed the search.

While the above invention has been described with reference to certain preferred embodiments, the scope of the present invention is not limited to these embodiments. One skilled in the art may find variations of these preferred embodiments which, nevertheless, fall within the spirit of the present invention, whose scope is defined by the claims set forth below.

Express Mail No. EV339025705US

8 References

[1] Keeney, Ralph L., Raiffa, Howard, *Decisions with Multiple Objectives*, Cambridge University Press:Cambridge, 1993.

[2] Saaty, Thomas L., *The Analytic Network Process*, RWS Publications:Pittsburg, 1996.

[3] A concise introduction to discrete choice analysis with a comparison to conjoint analysis is found at `http://www.action-research.com/discrete.htm`.

[4] Nesterov, Yurii, Nemirovskii, Arkadii, *Interior Point Polynomial Algorithms in Convex Programming*, Society for Industrial and Applied Math:Philadelphia, 1994.

[5] Tsang, Edward, *Foundations of Constraint Satisfaction*, Academic Press:London, 1993.

Petsec 35926

9 Abstract

A system is described which allows buyers to define their preferences and sellers to define their capabilities, then determines which trading points maximize the utility of the buyer. The system suggests trades by exploiting the flexibilities and tradeoffs encoded by both parties, thus providing win-win trades. A second level of optimization ranks the trades with all suppliers, allowing the buyer to rapidly determine the best alternatives. The system allows for rich negotiation spaces and supports continuous, discrete, and range or interval decision factors.

Document ID: 0.7.2.5.3.2.2

A Distance Function for Range Variables

The distance between buyer range variable $r_i = (\underline{r}_i, \overline{r}_i)$ and seller range variable $r_j = (\underline{r}_j, \overline{r}_j)$ is related to the overlap defined as

$$\text{overlap}(r_i, r_j) = \int_{\mu_j - \sigma_j}^{\mu_j + \sigma_j} dx\, \mathcal{N}(x; r_i)\mathcal{N}(x; r_j)$$

where $\mathcal{N}(x; r_i)$ is a Gaussian in x centered at $\mu_i = (\underline{r}_i + \overline{r}_i)/2$ with standard deviation $\sigma_i = \alpha(\overline{r}_i - \underline{r}_i)$ and α is a tunable parameter. The range of integration is bounded by the seller's capabilities. Evaluating the integral, we find

$$\text{overlap}(r_i, r_j) = \frac{1}{2\sqrt{2\pi(\sigma_i^2 + \sigma_j^2)}} \exp\left[-\frac{(\mu_i - \mu_j)^2}{2(\sigma_i^2 + \sigma_j^2)}\right] \left(\text{erf}\left[\frac{\sigma_j^2 + \sigma_j\mu_i - \sigma_j\mu_j + \sigma_i^2}{\sigma_i\sqrt{2(\sigma_i^2 + \sigma_j^2)}}\right] - \text{erf}\left[\frac{-\sigma_j^2 + \sigma_j\mu_i - \sigma_j\mu_j - \sigma_i^2}{\sigma_i\sqrt{2(\sigma_i^2 + \sigma_j^2)}}\right]\right).$$

By adjusting α we can control the hardness of the boundary at the ends of the range (smaller α corresponds to sharper dropoff outside the range). The maximum overlap will occur when $\mu_j = \mu_i$ and $\sigma_j = 0$. Substituting those values in the above equation, we get

$$\text{maxOverlap} = \frac{\text{erf}\left[\frac{1}{\sqrt{2}}\right]}{\sqrt{2\pi(\sigma_i^2 + \sigma_j^2)}}$$

The distance $d(r_i, r_j)$ is related to the overlap so that it varies from 0 (for identical range variables) to ∞ (for maximally distant range variables). Consequently, we define

$$R(r_i, r_j) = -\ln\left[\frac{\text{overlap}(r_i, r_j)}{\text{maxOverlap}}\right].$$

B From Rankings to Numerical Values

The numerical values in the tables $Z(\varkappa_i, \varkappa_j)$ can be formed from the buyer's rankings in a number of ways. Since we are minimizing distance, we want preferred options to contribute little Z and progressively less favorable options to have progressively larger Z values. To approach this, we first consider the problem of defining a Z' value normalized to lie between 0 and 1 and then take $Z = -\ln[1 - Z']$.

One possibility for Z' is a linear scaling. If d_i is the integer indicating the number of values in $\mathcal{D}_i$, $r_i^{(\alpha)}(\varkappa_j)$ indicates the buyer-specified rank (smaller rank means more preferable) for the option $\varkappa_i = \alpha \in \mathcal{D}_i$ conditional on $\varkappa_j$, then we might set $Z'(\varkappa_i = \alpha, \varkappa_j) = (r_i^{(\alpha)}(v_j) - 1)/(d_i - 1)$. Alternatively, we might use a non-linear scaling and set $Z(\varkappa_i = \alpha, v_j) = (1 - \beta^{r_i^{(\alpha)}(\varkappa_j)-1})/(1 - \beta^{d_i - 1})$ for some $\beta \in [0, 1]$ (exponential) or set $Z(v_i = \alpha, v_j) = \big((r_i^{(\alpha)}(v_j) -$

type	Z'
linear	$\ln\left[\dfrac{R_i-1}{R_i-r_i^{(\alpha)}(v_j)}\right]$
exponential	$\ln\left[\dfrac{1-\beta^{R_i-1}}{\beta^{r_i^{(\alpha)}(v_j)-1}-\beta^{R_i-1}}\right]$
algebraic	$-\ln\left[1-\left(\dfrac{r_i^{(\alpha)}(v_j)-1}{R_i-1}\right)^n\right]$

Table 4: Possible numerical ranking functions.

$1)/(R_i-1))^n$ for some $n>0$ (algebraic). The corresponding scaling functions for Z are given in Table 4.

For those allowed values of $\varkappa_i$ which are left unranked by the buyer, we can assign them a common numerical value greater than the lowest ranked value. For example if $r_i(\varkappa_j)$ is the largest ranked value we might assign all remaining $\varkappa_i$ values to have $Z(\varkappa_i,\varkappa_j)=r_i(\varkappa_j)/(d_i-1)$ in the linear case or $Z(\varkappa_i,\varkappa_j)=(1-\beta^{r_i(\varkappa_j)})/(1-\beta^{d_i-1})$ or $Z(\varkappa_i,\varkappa_j)=(1-1/(r_i(\varkappa_j)+1))/(1-1/d_i)$ in the non-linear cases.

C Determination of Scaling Factors

In this section we supply the details necessary to determine the scaling factors Q_r and Q_d. In the equations defining the average distances we have need of the integrals $\int_V d\mathbf{u}\ \exp[-(\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})]$ and $\int_V d\mathbf{u}\ (\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})\exp[-(\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})]$. In general, these integrations are not analytically tractable; however we will assume that the region of integration defined by $\overline{x}$ and $\underline{x}$ extends at least 2 standard deviations out in all directions so that the volume of integration is effectively infinite. In this case we can evaluate the integrals in closed form:

$$\int_V d\mathbf{u}\ \exp[-(\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})] \approx \sqrt{\frac{\pi^{n_c}}{\det \mathbf{C}^{-1}}}$$

and

$$\int_V d\mathbf{u}\ (\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})\exp[-(\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})] \approx \frac{n_c}{2}\sqrt{\frac{\pi^{n_c}}{\det \mathbf{C}^{-1}}}.$$

To this point we have not been specific about what is meant by the average over the range variables. In fact, we will assume that, however we define this average, it is independent of what the buyer requested and thus can be removed from inside the sum over the discrete variables. In this case we will concentrate on setting the discrete and continuous components equal. We have $\langle d_c\rangle/n_c=\langle d_d\rangle/n_d$ where

$$\frac{\langle d_d\rangle}{n_d}\equiv\frac{Q_d}{n_d}\frac{\sum_{\mathbf{v}} d_d\exp[-Q_d d_d]\int_V d\mathbf{u}\ \exp[-d_c]}{\sum_{\mathbf{v}}\exp[-Q_d d_d]\int_V d\mathbf{u}\ \exp[-d_c]} \quad\text{and}\quad \frac{\langle d_c\rangle}{n_c}\equiv\frac{1}{n_c}\frac{\sum_{\mathbf{v}}\exp[-Q_d d_d]\int_V d\mathbf{u}\ d_c\exp[-d_c]}{\sum_{\mathbf{v}}\exp[-Q_d d_d]\int_V d\mathbf{u}\ \exp[-d_c]}$$

and cancelling denominators Q_d is determined by solving

$$\frac{Q_d}{n_d}\sum_{\mathbf{v}} d_d \exp[-Q_d d_d]\int_V d\mathbf{u}\ \exp[-d_c] = \frac{1}{n_c}\sum_{\mathbf{v}} \exp[-Q_d d_d]\int_V d\mathbf{u}\ d_c \exp[-d_c].$$

This can be simplified further given our assumptions about the Gaussian integral:

$$\frac{Q_d}{n_d}\sum_{\mathbf{v}} \frac{d_d \exp[-Q_d d_d]}{\sqrt{\det \mathbf{C}^{-1}(\mathbf{v})}} = \frac{1}{2}\sum_{\mathbf{v}} \frac{\exp[-Q_d d_d]}{\sqrt{\det \mathbf{C}^{-1}(\mathbf{v})}}.$$

In most cases there will also be at least one finite root, which is what we need. If there are a multiplicity of roots, we select that root having the smallest value to ensure that the distribution defined by the utility function is as flat as possible. If there are no real roots, then we select the Q_d which minimizes the error.

D Phase 1 Optimization

In this section we detail the solution to the phase 1 optimization problem defined in Eq. (5).

D.1 General Formulation

We begin by writing down the objective function:

$$d_1(\mathbf{x}, \boldsymbol{\varkappa}) = \left(\mathbf{x}^{(s)} - \boldsymbol{\mu}(\boldsymbol{\varkappa})\right)^t \mathbf{C}_w^{-1}\left(\mathbf{x}^{(s)} - \boldsymbol{\mu}(\boldsymbol{\varkappa})\right) + R(\mathbf{r}; \boldsymbol{\varkappa})$$

where (see Eq. (15))

$$\mathbf{x}^{(s)} = \left(\mathbf{I} - \mathbf{M}^{(s)}(\boldsymbol{\varkappa}^{(s)}, \{b_{i,k}^{(s)}\})\right)^{-1}\mathbf{c}(\boldsymbol{\varkappa}^{(s)}, \{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\}) + \left(\mathbf{I} - \mathbf{M}^{(s)}(\mathbf{v}^{(s)}, \{b_{i,k}^{(s)}\})\right)^{-1}\mathbf{M}^{(b)}(\{b_{i,k}^{(b)}\})\mathbf{u}^{(b)} \tag{15}$$

as long as $x_k^{(s)}(b_{i,k}^{(s)}) \le x_k^{(s)} \le x_k^{(s)}(b_{i,k}^{(s)} + 1)$. This must be optimized over $\mathbf{x}$ subject to the constraints $\mathbf{G}_1(\boldsymbol{\varkappa})\mathbf{x}^{(s)} = \mathbf{g}_1(\boldsymbol{\varkappa})$ and $\mathbf{G}_2(\boldsymbol{\varkappa})\mathbf{x}^{(s)} = \mathbf{g}_2(\boldsymbol{\varkappa})$ for a given $\boldsymbol{\varkappa}$. Since $\boldsymbol{\varkappa}$ is fixed throughout the optimization $R(\mathbf{r}; \boldsymbol{\varkappa})$ is just a constant which can be evaluated and then ignored as far as determination of $\mathbf{x}$ is concerned. We simplify our notation so that $d_1(\mathbf{u}) = \left(\mathbf{x}^{(s)} - \boldsymbol{\mu}\right)^t \mathbf{C}_w^{-1}\left(\mathbf{x}^{(s)} - \boldsymbol{\mu}\right) + R(\mathbf{r})$. Similarly,

$$\begin{aligned}\mathbf{x}^{(s)} &= \left(\mathbf{I} - \mathbf{M}^{(s)}(\{b_{i,k}^{(s)}\})\right)^{-1}\mathbf{c}(\{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\}) + \left(\mathbf{I} - \mathbf{M}^{(s)}(\{b_{i,k}^{(s)}\})\right)^{-1}\mathbf{M}^{(b)}(\{b_{i,k}^{(b)}\})\mathbf{x}^{(b)} \\ &\equiv \mathbf{s}(\{b_{i,k}^{(s)}\}, \{b_{i,k}^{(b)}\}) + \mathbf{S}(\{b_{i,k}^{(s)}, \{b_{i,k}^{(b)}\})\mathbf{x}^{(b)}\end{aligned}$$

where we have defined the n_c-vector $\mathbf{s} = (\mathbf{I} - \mathbf{M}^{(s)})^{-1}\mathbf{c}$ and the $n_c \times n_c'$ matrix $\mathbf{S} = (\mathbf{I} - \mathbf{M}^{(s)})^{-1}\mathbf{M}^{(b)}$.[28] $n_c' < n_c$ is the number of continuous dimensions upon which the supplier response depends. Putting these pieces together we have

$$d_1(\mathbf{x}^{(b)}) = \mathbf{x}^{(b)t}\mathbf{Q}(\{b_{i,k}^{(s)}, \{b_{i,k}^{(b)}\})\mathbf{x}^{(b)} + \mathbf{q}^t\mathbf{x}^{(b)} + const \tag{16}$$

[28] In the next section we consider $\mathbf{S}$ further, taking advantage of any redundant variables to simplify the QP task.

where $\mathbf{Q}$ is an $n'_c \times n'_c$ matrix given by

$$\mathbf{Q}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\}) = \mathbf{S}^t(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\})\mathbf{C}_w^{-1}\mathbf{S}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\})$$

$\mathbf{q}$ is an n'-vector given by

$$\mathbf{q}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\}) = 2\mathbf{S}^t(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\})\mathbf{C}_w^{-1}\big(\mathbf{s}\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\}) - \boldsymbol{\mu}\big)^t.$$

and the constant term is

$$const = \big(\mathbf{s}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\}) - -\boldsymbol{\mu}\big)^t\mathbf{C}_w^{-1}\big(\mathbf{s}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\}) - \boldsymbol{\mu}\big) + R(\mathbf{r})$$

It is easily shown that the matrix $\mathbf{Q}$ is positive semidefinite. $d_1(\mathbf{u}^{(b)})$ must be optimized subject to the constraints

$$\mathbf{G}_1\mathbf{S}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\})\mathbf{x}^{(b)} = \mathbf{g}_1 - \mathbf{G}_1\mathbf{s}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\}), \tag{17}$$

$$\mathbf{G}_2\mathbf{S}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\})\mathbf{x}^{(b)} \leq \mathbf{g}_2 - \mathbf{G}_2\mathbf{s}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\}), \tag{18}$$

$$\underline{\mathbf{x}}^{(s)}(i) \leq \mathbf{s}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\}) + \mathbf{S}(\{b^{(s)}_{i,k}, \{b^{(b)}_{i,k}\})\mathbf{x}^{(b)} \leq \overline{\mathbf{x}}^{(s)}(i) \;\; \forall\, i, \tag{19}$$

$$\underline{\mathbf{x}}^{(b)}(i) \leq \mathbf{x}^{(b)} \leq \overline{\mathbf{x}}^{(b)}(i) \;\; \forall\, i \tag{20}$$

where $\underline{\mathbf{x}}^{(s)}(i)^t = \begin{bmatrix} x_1(b^{(s)}_{i,1}) & \cdots & x_n(b^{(s)}_{i,n}) \end{bmatrix}$ and $\overline{\mathbf{x}}^{(s)}(i)^t = \begin{bmatrix} x_1(b^{(s)}_{i,1}+1) & \cdots & x_n(b^{(s)}_{i,n}+1) \end{bmatrix}$ are the vectors of breakpoints for the $f^{(s)}_{i,k}$ $(k \in [1, n_c])$ functions. Similarly, for the buyer variables, $\underline{\mathbf{x}}^{(b)}(i)^t = \begin{bmatrix} x_1(b^{(b)}_{i,1}) & \cdots & x_n(b^{(b)}_{i,n}) \end{bmatrix}$ and $\overline{\mathbf{x}}^{(b)}(i)^t = \begin{bmatrix} x_1(b^{(b)}_{i,1}+1) & \cdots & x_n(b^{(b)}_{i,n}+1) \end{bmatrix}$ are the vectors of breakpoints for the $f^{(b)}_{i,k}$ $(k \in [1, n_c])$ functions.

d_1 is optimized in two steps. First we assume a value for the integer variables $\{b^{(s)}_{i,k}\}$ and $\{b^{(b)}_{i,k}\}$ fixing the constraints in Eqs. (17) and (18). We then minimize the quadratic d_1 subject to these constraints using a standard quadratic programming (QP) algorithm[29] to find $\mathbf{x}^{(b)}_{\text{opt}}(\{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\})$. This result is then used to find $\mathbf{x}^*\big(\boldsymbol{\varkappa}; \{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\}\big) = \mathbf{s}\big(\{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\}\big) + \mathbf{S}\big(\{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\}\big)\mathbf{x}^{(b)}_{\text{opt}}\big(\{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\}\big)$. By searching exhaustively over $\{b^{(s)}_{i,k}\}$ and $\{b^{(b)}_{i,k}\}$ we determine

$$\mathbf{x}^{(s)}(\boldsymbol{\varkappa}) = \underset{\{b^{(s)}_{i,k}\},\{b^{(b)}_{i,k}\}}{\arg\min}\; d_1\big(\mathbf{x}^*\big(\boldsymbol{\varkappa}; \{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\}\big), \boldsymbol{\varkappa}\big)$$

Before turning to the manner in which this optimization is accomplished in section D.3, we discuss a few improvements to the input into the QP algorithm.

D.2 The Quadratic Program

In this section we discuss improvements in the specification of the quadratic programming task to make it better conditioned and more efficient.

It is possible that the matrix $\mathbf{Q}$ used as input to the QP is singular or near singular,

[29] Numerous third parties supply quadratic programming packages, some based on interior-point methods which are appropriate, in the present context where matrix $\mathbf{Q}$ may be singular.

depending on the rank of $\mathbf{S}_1$. Consequently, it is important to use an interior point QP algorithm for which this is not a problem.

D.2.1 Variable Elimination

There may be rows or columns of $\mathbf{S}$ which are entirely zero. It is important to take advantage of this structure to eliminate unnecessary computation.

If row i is zero, this indicates that supplier variable $u_i^{(s)}$ is independent of any buyer variable and is simply equal to s_i. If there are any constraints involving only supplier variable i, then these can be checked before calling the QP solver to eliminate wasteful computation.[30] If the constraint is satisfied, we can eliminate it and then call the QP function. If column j of $\mathbf{S}$ is all zero, this means that there is no dependence of $\mathbf{u}^{(s)}$ on $u_j^{(b)}$ and so there is no need to optimize over that buyer variable. We will, however, still have to check that the constraints on $\mathbf{u}^{(s)}$ are satisfied. This observation allows us to simplify the quadratic programming problem by eliminating this unnecessary variable.

We formulate the problem to take advantage of both these observations. Assume there are $m < n_c$ non-zero rows of $\mathbf{S}$ and $m' < n_c'$ non-zero columns. Relabel the rows and columns so that

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$

where $\mathbf{S}_1$ is of size $m \times m'$ and, with a slight abuse of notation, the $\mathbf{0}$'s indicate matrices of the appropriate size. In like fashion we write

$$\mathbf{x}^{(s)} = \begin{bmatrix} \mathbf{x}_1^{(s)} \\ \mathbf{x}_2^{(s)} \end{bmatrix}, \quad \mathbf{x}^{(b)} = \begin{bmatrix} \mathbf{x}_1^{(b)} \\ \mathbf{x}_2^{(b)} \end{bmatrix}$$

where $\mathbf{x}_1^{(s)}$ is an m vector, $\mathbf{x}_2^{(s)}$ is an $n_c - m$ vector, $\mathbf{x}_1^{(b)}$ is an m' vector, and $\mathbf{x}_2^{(b)}$ is an $n_c' - m'$ vector. $\mathbf{x}_1^{(s)}$ indicates those components of $\mathbf{x}^{(s)}$ which vary with buyer variables while $\mathbf{x}_2^{(s)}$ indicates those variables which are constant. $\mathbf{x}_1^{(b)}$ are those buyer variables which affect at least one supplier variable and $\mathbf{x}_2^{(b)}$ are those buyer variables which do not affect any supplier variables. If we also partition $\mathbf{s}$ as $\mathbf{s}^t = \begin{bmatrix} \mathbf{s}_1^t & \mathbf{s}_2^t \end{bmatrix}$ then

$$\mathbf{x}_1^{(s)} = \mathbf{s}_1 + \mathbf{S}_1 \mathbf{x}_1^{(b)} \quad \text{and} \quad \mathbf{x}_2^{(s)} = \mathbf{s}_2.$$

Finally, we will also have need of the partitioned covariance matrix

$$\mathbf{C}^{-1} = \begin{bmatrix} \mathbf{C}_{1,1}^{-1} & \mathbf{C}_{1,2}^{-1} \\ \mathbf{C}_{2,1}^{-1} & \mathbf{C}_{2,2}^{-1} \end{bmatrix}$$

where $(\mathbf{C}_{1,2;w}^{-1})^t = \mathbf{C}_{2,1;w}^{-1}$ since $\mathbf{C}_w^{-1}$ is symmetric.

[30] Additionally, the QP doesn't like to have a row of zeros in the constraint matrix.

Including the dependencies on μ, the quadratic form we must optimize is

$$\left[(\mathbf{s}_1 - \mu_1)^t + (\mathbf{x}_1^{(b)})^t \mathbf{S}_1^t \quad (\mathbf{s}_2 - \mu_2)^t\right] \begin{bmatrix} \mathbf{W}_1\mathbf{C}_{1,1}^{-1}\mathbf{W}_1 & \mathbf{W}_1\mathbf{C}_{1,2}^{-1}\mathbf{W}_2 \\ \mathbf{W}_2\mathbf{C}_{2,1}^{-1}\mathbf{W}_1 & \mathbf{W}_2\mathbf{C}_{2,2}^{-1}\mathbf{W}_2 \end{bmatrix} \begin{bmatrix} \mathbf{s}_1 - \mu_1 + \mathbf{S}_1\mathbf{u}_1^{(b)} \\ \mathbf{s}_2 - \mu_2 \end{bmatrix}$$

where we have made the obvious definitions

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_2 \end{bmatrix} \quad \text{and} \quad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}.$$

Expanding the quadratic, the constant term is

$$(\mathbf{s}_1-\mu_1)^t\mathbf{W}_1\mathbf{C}_{1,1}^{-1}\mathbf{W}_1(\mathbf{s}_1-\mu_1)+(\mathbf{s}_2-\mu_2)^t\mathbf{W}_2\mathbf{C}_{2,2}^{-1}\mathbf{W}_2(\mathbf{s}_2-\mu_2)+2(\mathbf{s}_1-\mu_1)^t\mathbf{W}_1\mathbf{C}_{1,2}^{-1}\mathbf{W}_2(\mathbf{s}_2-\mu_2).$$

The linear term is

$$\mathbf{q} = \left((\mathbf{s}_1 - \mu_1)^t\mathbf{W}_1\mathbf{C}_{1,1}^{-1}\mathbf{W}_1\mathbf{S}_1^t + 2(\mathbf{s}_2 - \mu_2)^t\mathbf{W}_2\mathbf{C}_{2,1}^{-1}\mathbf{W}_1\mathbf{S}_1\right)\mathbf{x}_1^{(b)}$$

and the quadratic term is

$$\mathbf{Q} = (\mathbf{x}_1^{(b)})^t\mathbf{S}_1^t\mathbf{W}_1\mathbf{C}_{1,1}^{-1}\mathbf{W}_1\mathbf{S}_1\mathbf{x}_1^{(b)}.$$

The constraints involving the optimization variables $\mathbf{u}_1^{(b)}$ are

$$\begin{aligned} \mathbf{G}_{1;1}\mathbf{S}_1\mathbf{x}_1^{(b)} &= \mathbf{g}_1 - \mathbf{G}_{1;1}\mathbf{s}_1 - \mathbf{G}_{1;2}\mathbf{s}_2, \\ \mathbf{G}_{2;1}\mathbf{S}_1\mathbf{x}_1^{(b)} &\le \mathbf{g}_2 - \mathbf{G}_{2;1}\mathbf{s}_1 - \mathbf{G}_{2;2}\mathbf{s}_2, \\ \underline{\mathbf{x}}_1^{(s)}(i) \le \mathbf{s}_1 + \mathbf{S}_1\mathbf{x}_1^{(b)} &\le \overline{\mathbf{x}}_1^{(s)}(i) \ \ \forall\, i, \\ \underline{\mathbf{x}}_1^{(b)}(i) \le \mathbf{x}_1^{(b)} &\le \overline{\mathbf{x}}_1^{(b)}(i) \ \ \forall\, i \end{aligned}$$

where we have simplified notation by dropping the dependence on breakpoints and defined

$$\mathbf{G}_i = \begin{bmatrix} \mathbf{G}_{i;1} & \mathbf{G}_{i;2} \end{bmatrix}$$

i.e., $\mathbf{G}_{i;1}$ is the columns of $\mathbf{G}_1$ involving the active buyer variables while $\mathbf{G}_{i;2}$ is formed from the inactive buyer variables. Similarly, the constraints which can be checked prior to even solving the QP are

$$\underline{\mathbf{x}}_2^{(s)}(i) \le \mathbf{s}_2 \le \overline{\mathbf{x}}_2^{(s)}(i) \ \ \forall\, i.$$

D.2.2 QP Normalization

There are two types of normalizations we perform for increased numerical stability. If the scales of the continuous dimensions vary greatly (i.e., one dimension has an ideal value of 10^{-2} and another dimension has ideal value 10^6), then the matrix $\mathbf{C}^{-1}$ becomes badly conditioned, which makes $\mathbf{S}$ badly conditioned and thus makes the QP unstable. Similarly, if the breakpoints have

radically, different lengths, the QP problem can also become unstable. We consider how to remedy each of these problems.

Dimension normalization

The quadratic form to be minimized is $(\mathbf{x}-\boldsymbol{\mu})^t\mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu})$. We modify this problem by defining the dimensionless variables ϕ_i through $x_i = \phi_i\mu_i$ in terms of which

$$(\mathbf{x}-\boldsymbol{\mu})^t\mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu}) = (\boldsymbol{\phi}-\mathbf{1})^t\boldsymbol{\mathcal{C}}^{-1}(\boldsymbol{\phi}-\mathbf{1})$$

where $\boldsymbol{\mathcal{C}}^{-1} = \mathbf{D}\mathbf{C}^{-1}\mathbf{D}$ and $\mathbf{D} = \mathrm{diag}(\boldsymbol{\mu})$. When making this variable transformation we must assure that the constraints on $\mathbf{x}$ are adjusted so that $\mathbf{G}_1\mathbf{x} = \mathbf{g}_1$ and $\mathbf{G}_2\mathbf{x} \leq \mathbf{g}_2$ become the constraints $\boldsymbol{\mathcal{G}}_1\boldsymbol{\phi} = \mathbf{g}_1$ and $\boldsymbol{\mathcal{G}}_2\boldsymbol{\phi} \leq \mathbf{g}_2$ on $\boldsymbol{\phi}$ where $\boldsymbol{\mathcal{G}}_{1,2} = \mathbf{G}_{1,2}\mathbf{D}$. If there are global bounds of the form $\underline{\mathbf{x}} \leq \mathbf{x} \leq \overline{\mathbf{x}}$, then these are translated to $\underline{\boldsymbol{\phi}} \leq \boldsymbol{\phi} \leq \overline{\boldsymbol{\phi}}$ where $\underline{\boldsymbol{\phi}} = \mathbf{D}^{-1}\underline{\mathbf{u}}$ and $\overline{\boldsymbol{\phi}} = \mathbf{D}^{-1}\overline{\mathbf{u}}$.

In addition to these changes, we also need to modify the capabilities appropriately. Recall Eq. (2) which reads (after dropping the dependence on the discrete variables)

$$x_i^{(s)} = \sum_k f_{i,k}^{(s)}(x_k^{(s)}) + f_{i,k}^{(b)}(x_k^{(b)}). \tag{21}$$

In terms of the dimensionless variables we have

$$\phi_i^{(s)} = \sum_k \frac{f_{i,k}^{(s)}(\phi_k^{(s)}\mu_k)}{\mu_i} + \frac{f_{i,k}^{(b)}(\phi_k^{(b)}\mu_k)}{\mu_i}. \tag{22}$$

Thus, when we use the dimensionless variables, the function $f_{i,k}$ needs to be modified so that the y dimension is divided by μ_i and the x dimension is multiplied by μ_k.

Once the optimization has determined an optimal $\boldsymbol{\phi}$, we return to the original variables through $\mathbf{x} = \mathbf{D}\boldsymbol{\phi}$.

Breakpoint range normalization

For larger problems it is sometimes important to normalize the input variables for the QP solver. This is carried out in order to improve the QP search performance in situations where there is large variation in the range of the segments $\phi_k^{(b)}(b_k) \leq \phi_k^{(b)} \leq \phi_k^{(b)}(b_k+1)$ and the magnitude of their values. This can be accomplished by introducing the normalization variables $\hat{\phi}_k^{(b)} = (\phi_k^{(b)} - \phi_k^{(b)}(b_k))/(\phi_k^{(b)}(b_k+1) - \phi_k^{(b)}(b_k))$. Here $\hat{\phi}_k^{(b)}$ will have minimum value of zero and maximum of one. We can now express $\phi^{(b)}$ as

$$\boldsymbol{\phi}^{(b)} = \boldsymbol{\Lambda}\hat{\boldsymbol{\phi}}^{(b)} + \mathbf{p} \tag{23}$$

where $\boldsymbol{\Lambda}$ is the diagonal matrix $\boldsymbol{\Lambda} = \mathrm{diag}(\phi_1^{(b_1+1)} - \phi_1^{(b_1)}, \phi_2^{(b_2+1)} - \phi_2^{(b_2)}, \cdots, \phi_m^{(b_m+1)} - \phi_m^{(b_m)})$ and $\mathbf{p}^t = [\phi_1^{(b_1)}, \phi_2^{(b_2)}, \cdots, \phi_m^{(b_m)}]$.

Expanding Eq.(16) using the definition of $\boldsymbol{\phi}$ in Eq.(23), we obtain

$$d_1(\mathbf{w};\{b_{i,k}\}) = \frac{1}{2}\hat{\boldsymbol{\phi}}^{(b)t}\boldsymbol{\Lambda}\mathbf{D}\mathbf{Q}(\{b_{i,k}\})\mathbf{D}\boldsymbol{\Lambda}\hat{\boldsymbol{\phi}}^{(b)} + \mathbf{q}^t(\{b_{i,k}\})\mathbf{D}\boldsymbol{\Lambda}\hat{\boldsymbol{\phi}}^{b} + \mathbf{p}^t\mathbf{D}\mathbf{Q}(\{b_{i,k}\})\mathbf{D}\boldsymbol{\Lambda}\hat{\boldsymbol{\phi}}^{(b)} + \frac{1}{2}\mathbf{p}^t\mathbf{D}\mathbf{Q}(\{b_{i,k}\})\mathbf{D}\mathbf{p} + \mathbf{q}^t(\{b_{i,k}\})\mathbf{D}\mathbf{p}.$$

Hence the quadratic programming problem to be solved in terms of $\hat{\boldsymbol{\phi}}$ is

$$QP(\hat{\mathbf{x}}^{(b)};\{b_{i,k}\}) = \frac{1}{2}\hat{\mathbf{x}}^{(b)t}\hat{\mathbf{Q}}(\{b_{i,k}\})\hat{\mathbf{x}}^{(b)} + \hat{\mathbf{q}}^t(\{b_{i,k}\})\hat{\mathbf{u}}^{(b)}, \tag{24}$$

where $\hat{\mathbf{Q}} = \boldsymbol{\Lambda}\mathbf{D}\mathbf{Q}(\{b_{i,k}\})\mathbf{D}\boldsymbol{\Lambda}$ and $\hat{\mathbf{q}}^t = \big(\mathbf{q}^t(\{b_{i,k}\}) + \mathbf{p}^t\mathbf{D}\mathbf{Q}(\{b_{i,k}\})\big)\mathbf{D}\boldsymbol{\Lambda}$. Similarly, the equality and inequality constraints of Eqs. (17) and (18) expressed in terms of $\hat{\boldsymbol{\phi}}$ are

$$\hat{\mathbf{G}}_1(\{b_{i,k}\})\hat{\boldsymbol{\phi}}^{(b)} = \hat{\mathbf{g}}_1 \quad \text{and} \quad \hat{\mathbf{G}}_2(\{b_{i,k}\})\hat{\boldsymbol{\phi}}^{(b)} \leq \hat{\mathbf{g}}_2, \tag{25}$$

where

$$\begin{aligned}
\hat{\mathbf{G}}_1(\{b_{i,k}\}) &= \mathbf{G}_1\mathbf{S}(\{b_{i,k}\})\mathbf{D}\boldsymbol{\Lambda}, \\
\hat{\mathbf{G}}_2(\{b_{i,k}\}) &= \mathbf{G}_2\mathbf{S}(\{b_{i,k}\})\mathbf{D}\boldsymbol{\Lambda}, \\
\hat{\mathbf{g}}_1 &= \mathbf{g}_1 - \mathbf{G}_1\mathbf{s}(\{b_{i,k}\}) - \mathbf{G}_1\mathbf{S}(\{b_{i,k}\})\mathbf{D}\mathbf{p}, \\
\hat{\mathbf{g}}_2 &= \mathbf{g}_2 - \mathbf{G}_2\mathbf{s}(\{b_{i,k}\}) - \mathbf{G}_1\mathbf{S}(\{b_{i,k}\})\mathbf{D}\mathbf{p}.
\end{aligned}$$

D.3 Optimization Over The Breakpoint Variables

We have detailed how to determine $\mathbf{x}^{(b)}_{\text{opt}}(\boldsymbol{\varkappa}, \{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\})$ given a specification of all $\{b^{(s)}_{i,k}\}$ and $\{b^{(b)}_{i,k}\}$, but how do we determine the best feasible choice of $\{b^{(s)}_{i,k}\}$ and $\{b^{(b)}_{i,k}\}$? Before we answer that question, we have to examine the infeasibilities that might arise due to improper choices of $\{b^{(s)}_{i,k}\}$ and $\{b^{(b)}_{i,k}\}$.

D.3.1 Elimination of $\{b_{i,k}\}$ Infeasibilities

Most of the settings of $\{b^{(b)}_{i,k}\}$ or $\{b^{(s)}_{i,k}\}$ will not yield a feasible solution because Eq. (20) or Eq. (19) will be violated. The reason is that, for any k, it is highly likely to get two segments $x^{(b)}_k(b_{i,k}) \leq x^{(b)}_k \leq x^{(b)}_k(b_{i,k}+1)$ and $x^{(b)}_k(b_{j,k}) \leq x^{(b)}_k \leq x^{(b)}_k(b_{j,k}+1)$ that do not overlap (i.e., the optimizer will not be able to find a $x^{(b)}_k$ value that satisfies both bounds, and hence no feasible solution will be found). The same is also true for supplier response segment $x^{(s)}_k$.

To eliminate those infeasibilities, we construct a super set of breakpoints $\{\varpi^b_k\}$ such that

$$\varpi^{(b)}_k = \bigcup_{i=1}^{n}\{x^{(b)}_k(b^{(b)}_{i,k}=1), x^{(b)}_k(b^{(s)}_{i,k}=2), \cdots, x^{(b)}_k(b^{(b)}_{i,k}=\kappa^{(b)}_{i,k})\}$$

Hence each piecewise linear function $f_{i,k}$ will be described by a set of breakpoints $\varpi_k = \{x_k(1), \cdots, x_k(\kappa_k)\}$ and function values $f_{i,k}(x_k(1)), \cdots, f_{i,k}(x_k(\kappa_k))$. These function values

at the new breakpoints are linearly interpolated from the old breakpoint-function pairs. After this reduction we have $\kappa_k - 1$ different $b_k \in [1, \kappa_k - 1]$ values, which eliminates the need for constraint Eq. (20).

The continuous supplier variables and constraint Eq. (19) are more problematic. Suppressing any dependence on $\boldsymbol{\varkappa}$, we know

$$x_i^{(s)} = c_i(\{b_{i,1}^{(s)}, \cdots, b_{i,i-1}^{(s)}\}, \{b_{i,k}^{(b)}\}) + \sum_{k<i} m_{i,k}^{(s)}(b_{i,k}^{(s)}) x_k^{(s)} + \sum_k m_{i,k}^{(b)}(b_{i,k}^{(b)}) x_k^{(b)}.$$

If $\{b_{i,k}^{(b)}\}$ and $\{b_{i,1}^{(s)}, \cdots, b_{i,i-1}^{(s)}\}$ are known, then this expression allows us to bound the values $x_i^{(s)}$ may take on, and thus bound the interval in which the feasible $b_{i,i}^{(s)}$ lie. If we define the functions

$$\text{smax}(s,x,y) = \begin{cases} s\max(x,y) & \text{if } s \geq 0 \\ s\min(x,y) & \text{if } s < 0 \end{cases} \quad \text{and} \quad \text{smin}(s,x,y) = \begin{cases} s\min(x,y) & \text{if } s \geq 0 \\ s\max(x,y) & \text{if } s < 0 \end{cases}$$

then the minimum and maximum values that $x_i^{(s)}$ may take on are:

$$\begin{aligned} \min(x_i^{(s)}) = c_i(\{b_{i,1}^{(s)}, \cdots, b_{i,i-1}^{(s)}\}, \{b_{i,k}^{(b)}\}) &+ \sum_{k<i} \text{smin}\big(m_{i,k}^{(s)}(b_{i,k}^{(s)}), x_k^{(s)}(b_{i,k}^{(s)}), x_k^{(s)}(b_{i,k}^{(s)}+1)\big) \\ &+ \sum_k \text{smin}\big(m_{i,k}^{(b)}(b_{i,k}^{(b)}), x_k^{(b)}(b_{i,k}^{(b)}), x_k^{(b)}(b_{i,k}^{(b)}+1)\big), \end{aligned} \tag{26}$$

and

$$\begin{aligned} \max(x_i^{(s)}) = c_i(\{b_{i,1}^{(s)}, \cdots, b_{i,i-1}^{(s)}\}, \{b_{i,k}^{(b)}\}) &+ \sum_{k<i} \text{smax}\big(m_{i,k}^{(s)}(b_{i,k}^{(s)}), x_k^{(s)}(b_{i,k}^{(s)}), x_k^{(s)}(b_{i,k}^{(s)}+1)\big) \\ &+ \sum_k \text{smax}\big(m_{i,k}^{(b)}(b_{i,k}^{(b)}), x_k^{(b)}(b_{i,k}^{(b)}), x_k^{(b)}(b_{i,k}^{(b)}+1)\big) \end{aligned} \tag{27}$$

respectively. The above equations imply that we can fully translate a complete specification of buyer variable breakpoint intervals to a corresponding set of feasible supplier variable breakpoint intervals. We will search over the breakpoints using a branch-and-bound algorithm, and will need to be able to infer the restriction on the values of feasible supplier breakpoint intervals given a partial specification of buyer breakpoint interval variables. This inference, which narrows the domains of the supplier breakpoint variables, is called constraint propagation. Before describing how the branch-and-bound algorithm search over breakpoint interval variables is accomplished, we show how the constraint propagation may be obtained.

Narrowing of Supplier Breakpoint Domains Through Propagation

Let $\boldsymbol{\beta}_k^{(b)}$ be a vector of integers specifying a set of breakpoint intervals for the kth buyer variable. $|\boldsymbol{\beta}_k| = 1$ if the interval is uniquely determined, and $|\boldsymbol{\beta}_k| = \kappa_k^{(b)} - 1$ if no restrictions on the interval have been specified. We index the lth component of $\boldsymbol{\beta}_k^{(b)}$ as $\beta_{k;l}^{(b)}$, $l \in [1, \kappa_k^{(b)} - 1]$. A set of breakpoints for all buyer dimensions is written as $\{\boldsymbol{\beta}_k^{(b)}\}$. In analogous fashion, we define $\boldsymbol{\beta}_k^{(s)}$, $\beta_{k;l}^{(s)}$, $l \in [1, \kappa_k^{(s)} - 1]$, and $\{\boldsymbol{\beta}_k\}$ for the supplier breakpoint interval variables.

From a specification $\{\boldsymbol{\beta}_k^{(b)}\}$ we infer the feasible domains $\{\boldsymbol{\beta}_k^{(s)}\}$ using $\max(x_i^{(s)})$ and $\min(x_i^{(s)})$ where:

$$\max(x_i^{(s)}) = \sum_k \max_{b_k^{(b)} \in \boldsymbol{\beta}_k^{(b)}} \left\{ c_{i,k}(\{b_k^{(b)}\}) + \operatorname{smax}\left(m_{i,k}^{(b)}(b_k^{(b)}), x_k^{(b)}(b_k^{(b)}), x_k^{(b)}(b_k^{(b)}+1)\right) \right\} + \sum_{k<i} \max_{b_k^{(s)} \in \boldsymbol{\beta}_k^{(s)}} \left\{ c_{i,k}(\{b_k^{(s)}\}) + \operatorname{smax}\left(m_{i,k}^{(s)}(b_k^{(s)}), x_k^{(s)}(b_k^{(s)}), x_k^{(s)}(b_k^{(s)}+1)\right) \right\} \quad (28)$$

and

$$\min(x_i^{(s)}) = \sum_k \min_{b_k^{(b)} \in \boldsymbol{\beta}_k^{(b)}} \left\{ c_{i,k}(\{b_k^{(b)}\}) + \operatorname{smin}\left(m_{i,k}^{(b)}(b_k^{(b)}), x_k^{(b)}(b_k^{(b)}), x_k^{(b)}(b_k^{(b)}+1)\right) \right\} + \sum_{k<i} \min_{b_k^{(s)} \in \boldsymbol{\beta}_k^{(s)}} \left\{ c_{i,k}(\{b_k^{(s)}\}) + \operatorname{smin}\left(m_{i,k}^{(s)}(b_k^{(s)}), x_k^{(s)}(b_k^{(s)}), x_k^{(s)}(b_k^{(s)}+1)\right) \right\} \quad (29)$$

We begin with $i = 1$ (where no supplier variables contribute to $x_1^{(s)}$) and determine the interval $I_1 = [\min(x_1^{(s)}), \max(x_1^{(s)})]$. We then form the vector $\boldsymbol{\beta}_1^{(s)}$ with components $\{\beta_{1;l}^{(s)} | [x_1^{(s)}(\beta_{1;l}), x_1^{(s)}(\beta_{1;l}+1)] \cap I_1 \neq \emptyset\}$. Equations (28) and (29) can then be iterated to find the interval $I_2 = [\min(x_2^{(s)}), \max(x_2^{(s)})]$ using the previously calculated $\min(x_1^{(s)})$ and $\max(x_1^{(s)})$. With I_2 known, the components of $\boldsymbol{\beta}_2^{(s)}$ are determined as above. Iterating this procedure, we can find the potentially narrowed domains of all $\{\beta_k^{(s)}\}$.

We now return to the optimization problem: we need to search over all $\prod_k \kappa_k^{(b)} \times \prod_k \kappa_k^{(s)}$ combinations of breakpoints to determine the combination that minimizes $d_1\left(\mathbf{x}_{\text{opt}}(\boldsymbol{\varkappa}, \{b_k^{(b)}\}, \{b_k^{(s)}\}), \boldsymbol{\varkappa}\right)$. The simplest manner in which this accomplished is via exhaustive enumeration over all $\{b_k^{(b)}\}$ and $\{b_k^{(s)}\}$. However, once all $b_k^{(b)}$ have been specified, we need only search exhaustively over the corresponding $\{\beta_k(s)\}$ inferred as above. There are, however, many ways to speed up the search over the buyer breakpoint intervals. We describe two methods.

D.3.2 Local Search

This method is extremely fast but not guaranteed to return the global optimum across all buyer breakpoint intervals. Here, initial random and feasible[31] $\{b_k^{(s)}\}$, and $\{b_k^{(b)}\}$ sets are first created. After the continuous optimization is carried out using the initial $\{b_k^{(s)}\}$, and $\{b_k^{(b)}\}$, a new $\{\tilde{b}_k^{(s)}\}$, and $\{\tilde{b}_k^{(b)}\}$ set is generated from the old one by examining whether each generated $x_k^{(b)}(\{b_k^{(s)}\}, \{b_k^{(b)}\})$ solution is on the boundaries of the interval $x_k^{(b)}(b_k) \leq x_k^{(b)} \leq x_k^{(b)}(b_k+1)$ or $x_k^{(s)}(b_k) \leq x_k^{(s)} \leq x_k^{(s)}(b_k+1)$. If $x_k^{(b)}$ is on the lower bound of the interval, $\tilde{b}_k^{(b)} = b_k^{(b)} - 1$; similarly, if it is on the upper bound, $\tilde{b}_k^{(b)} = b_k^{(b)} + 1$. If $x_k^{(b)}$ is not on either end, then $\tilde{b}_k^{(b)} = b_k^{(b)}$. Of course this done while maintaining $\tilde{b}_k^{(b)}$ in the interval $1 \leq \tilde{b}_k^{(b)} \leq \kappa_k^{(b)} - 1$. Supplier breakpoint variables are updated in a similar manner. The procedure is repeated until no decrease in distance is attained.

[31] Feasible in the sense that they do not violate the constraints in equations (17), (18), (19), and (20).

D.3.3 Branch-and-Bound Search

Branch-and-bound is an efficient procedure used to exhaustively search a space and eliminate configurations that can be proven to be suboptimal. The theory behind branch-and-bound is well understood [5]. The idea is to structure the space as a tree of partially-completed possibilities and to explore the tree so that good complete solutions are constructed early on. Branches of the tree that can be shown to yield solutions which cannot be better than the currently best solution need not be explored. Heuristics indicate the order in which to explore the tree and provide a lower bound on the fitness of partially completed solutions for pruning of suboptimal branches..

In the present context, we carry out branch-and-bound search over the space of possible buyer breakpoint combinations. The tree is structured beginning from a node at which none of the $b_k^{(b)}$ have been assigned. From this node, the tree branches into the $\kappa_i^{(b)} - 1$ possible assignations for $b_i^{(b)}$. From any of these nodes, the tree branches into the $\kappa_j^{(b)}$ possible values for $b_j^{(b)}$. The tree branches at each node to assign the remaining variables. At the leaves of the tree, all buyer breakpoint variables have been assigned.

From any node, the search proceeds by choosing the most promising node to consider, as indicated by a variable ordering heuristic and considering the possible values for that variable in the order indicated by the value ordering heuristic. A complete solution and its accompanying distance is obtained when the search ends at a leaf of the tree. To avoid unnecessary exploration, a lower bound heuristic indicates when a partially completed solution (node) need not be expanded because all completions will have higher distance than the best located so far. We describe these three heuristics for search over buyer breakpoints.

Lower Bound

Given a set of buyer breakpoint interval vectors $\{\boldsymbol{\beta}_k^{(b)}\}$, we can propagate to $\{\boldsymbol{\beta}^{(s)}\}$ and the intervals $[\min(x_i^{(s)}), \max(x_i^{(s)})]$. A lower bound on the value d_1 can be obtained by ignoring the supplier capabilities and minimizing

$$d_1(\mathbf{x}) = (\mathbf{x}^{(s)} - \boldsymbol{\mu})^t \mathbf{C}_w^{-1} (\mathbf{x}^{(s)} - \boldsymbol{\mu}) \tag{30}$$

subject to

$$\mathbf{G}_1 \mathbf{x}^{(\mathbf{s})} = \mathbf{g}_1 \tag{31}$$

$$\mathbf{G}_2 \mathbf{x}^{(\mathbf{s})} \leq \mathbf{g}_2 \tag{32}$$

$$\min(x_i^{(s)}) \leq x_i^{(s)} \leq \max(x_i^{(s)}), \tag{33}$$

The lower bound thus obtained is denoted by $\mathrm{lb}(\{\boldsymbol{\beta}^{(b)}\})$.

Variable and Value Ordering

The lower bound $\mathrm{lb}(\{\boldsymbol{\beta}^{(b)}\})$ can be used to define an ordering over variables and the values any variable may take on. We define the functions $L_{k,l}$ which operate on the set $\{\boldsymbol{\beta}^{(b)}\}$ by locking the kth buyer breakpoint variable to $\beta_{k;l}$. At any node we can determine the lower

bounds $\mathrm{lb}(L_{k,l}\{\boldsymbol{\beta}^{(b)}\})$. We can select to explore the variables in order of increasing lb_k where

$$\mathrm{lb}_k(\{\boldsymbol{\beta}^{(b)}\}) = \frac{\sum_{l=1}^{|\boldsymbol{\beta}_k^{(b)}|} \mathrm{lb}(L_{k,l}(\{\boldsymbol{\beta}^{(b)}\}))}{\sum_{l=1}^{|\boldsymbol{\beta}_k^{(b)}|} 1},$$

i.e., the average over all possible completions for $b_k^{(b)}$.[32] Within any given variable the (say k^*) values can be explored in order of increasing $\mathrm{lb}(L_{k^*,l}(\{\boldsymbol{\beta}^{(b)}\}))$. If we are willing to do more computation, we can explore the branches in an interleaved order according to all the $\mathrm{lb}(L_{k,l}(\{\boldsymbol{\beta}^{(b)}\}))$.

Pruning and Search

During the branch-and-bound search, branches can be pruned if the node has $\mathrm{lb}(L_{k,l}(\{\boldsymbol{\beta}^{(b)}\}))$ greater than the distance of the lowest examined leaf node.

Search begins at the root node where each $\boldsymbol{\beta}_k^{(b)} = [1, \cdots, \kappa_k^{(b)} - 1]$. We applying the ordering heuristics above to move to a node where one breakpoint variable has been fixed. From that node, we move down the tree, fixing a value at a time until we reach a leaf. We then progressively backtrack up the tree, eliminating branches whose lower bounds are greater than the best leaf examined.

Once the optimal set of buyer breakpoints has been determined, we search exhaustively over the supplier breakpoints feasible with this choice according to Equations (26) and (27).

Improvements

We have not yet exploited the fact that, from any node having $\{\boldsymbol{\beta}_k^{(b)}\}$, the constraints in Equation (19) may narrow the domains of the buyer breakpoint variables deeper in the tree. The computational cost of narrowing the breakpoint domains may be warranted if the pruning thus obtained is sufficient.

E Phase 2 Optimization

In this section we detail an efficient algorithm to optimize Eq. (5):

$$\arg\min_{\boldsymbol{\varkappa}} \; d_1(\mathbf{x}(\boldsymbol{\varkappa}), \boldsymbol{\varkappa}) + d_2(\boldsymbol{\varkappa}) \quad \text{subject to} \quad \mathcal{C}^{(b)}(\boldsymbol{\varkappa}) \wedge \mathcal{C}^{(s)}(\boldsymbol{\varkappa}).$$

Our approach is a marriage of constraint programming and local search.

E.1 Local Search over D-neighborhoods

We begin with a definition. The d-neighbors of a discrete configuration $\boldsymbol{\varkappa}$ are all those configurations, $\boldsymbol{\varkappa}'$, which differ from $\boldsymbol{\varkappa}$ in d attributes. Attribute i differs in two configurations if $\varkappa_i \neq \varkappa_i'$. Thus the distance varies from 0 if the configurations are identical, up to n_d if all attributes are different. The D-neighborhood of $\boldsymbol{\varkappa}$ is the union of all $d = 1$ neighbors, $d = 2$

[32] We might also take the minimum over all possible completions $\mathrm{lb}_k(\{\boldsymbol{\beta}^{(b)}\}) = \min_{l \in [1, |\boldsymbol{\beta}_k^{(b)}|]} \mathrm{lb}(L_{k,l}(\{\boldsymbol{\beta}^{(b)}\}))$.

1	***11
2	**1*1
3	**11*
4	*1**1
5	*1*1*
6	*11**
7	1***1
8	1**1*
9	1*1**
10	11***

(a)

1	***+1
2	**1*1
3	**1+*
4	*+**1
5	*+*+*
6	*+1**
7	1***1
8	1**+*
9	1*1**
10	1+***

(b)

Table 5: Configurations examined in the $D = 3$ neighborhood search around (a) the initial configuration 11111 and (b) the initial configuration 0+0+0 (duplicate search configurations are indicated in bold) .

neighbors, up to $d = D$ neighbors. The D-neighborhood around $\boldsymbol{\varkappa}$ is indicated by $\mathcal{N}_D(\boldsymbol{\varkappa})$. We note that, for any configuration, its n_d neighborhood is the entire discrete space.

Using the D-neighborhood, we apply a standard hillclimbing algorithm which respects all the discrete constraints. D is specified before the algorithm begins and serves as an efficient manner to trade off running time with the quality of the final solution. The hillclimbing algorithm takes as input a starting configuration $\boldsymbol{\varkappa}_t$ and finds the configuration $\boldsymbol{\varkappa}_{t+1} \in \mathcal{N}_D(\boldsymbol{\varkappa}_t)$ which minimizes $d_1(\mathbf{x}(\boldsymbol{\varkappa}), \boldsymbol{\varkappa}) + d_2(\boldsymbol{\varkappa})$ subject to $C^{(b)}(\boldsymbol{\varkappa}) \wedge C^{(s)}(\boldsymbol{\varkappa})$. The algorithm then iterates again starting from $\boldsymbol{\varkappa}_{t+1}$. The hillclimbing algorithm terminates when it finds a configuration $\boldsymbol{\varkappa}_T$ which has a distance less than or equal to all of its D-neighbors. If $D = n_d$, this is the global minimum. Given a function which exhaustively searches a D-neighborhood for the lowest distance feasible configuration, a D-neighborhood hillclimbing algorithm is easily implemented.

There is one noteworthy point about the implementation of an efficient D-neighborhood hillclimber. Consider a $D = 3$ search over $n_d = 5$ discrete variables. Without loss of generality, we label the initial configuration as 11111. In the $D = 3$ neighborhood search, the configurations examined are given in Table 5(a) as indicated by the *s. Let us assume that the best neighbor in the D neighborhood of 11111 differs in components 2 and 4. We indicate this best neighbor as 1+1+1, which serves as the starting configuration for a new $D = 3$ search. The configurations examined in the second neighborhood search are given in Table 5(b). From Table 5(b) we see that the search branches 2, 7, and 10 are redundant, having already been performed in the D neighborhood search around 11111.[33] Thus these branches may be pruned in the second D neighborhood search. A search branch covers a previous search and may be pruned if the vector of positions of the wildcard characters covers the vector of positions at

[33]The duplication of search effort is most acute when D is large and the fittest neighbor is found very close by (small d).

which the previous best solution was found. The binary relation covers is defined as

$$\boldsymbol{\varkappa} \text{ covers } \boldsymbol{\varkappa}' \iff \varkappa_i' \in \boldsymbol{\varkappa} \ \forall \ \varkappa_i' \in \boldsymbol{\varkappa}'.$$

The covers relation is transitive but not reflexive. Thus, to increase the efficiency, we can keep a list of the previous branches searched and ignored search branches that have already been covered.

E.1.1 Improvements

Throughout the present discussion we have assumed that D remains constant through the hillclimbing process. There are good reasons why this should not be the case. With any reasonable landscape correlation structure, we know that lower distance neighbors are more likely to be found early in the hillclimbing process. Thus a reasonable heuristic is to start with a larger D early on and gradually lower it with time.

E.2 Exhaustive Search of a D-neighborhood

To complete the discussion of the phase 2 optimization, we describe a computationally efficient way in which to find the lowest distance feasible configuration within a D-neighborhood. We use a branch-and-bound method with heuristics tuned for the present problem. The factors tuned include the variable order in which we search, the order of the values any given variable may take, and a heuristic which says when certain branches of the search space may be safely ignored.

We use heuristics which early on in the search generate low-distance configurations, and later prune parts of the search that we can prove must have higher distance than the best configuration generated so far.

E.2.1 Heuristics And Lower Bounds

We focus on obtaining good heuristics for the discrete contribution to the total distance, then later incorporate the continuous and range contributions since these are simpler.

At any point in the search, some variables have been assigned and some have not. We divide up the sum over the $Z_{i,j}(\varkappa_i, \varkappa_j)$ into four contributions

$$\begin{aligned} Q_d Z_w(\boldsymbol{\varkappa}) = {} & Q_d \sum_{i \text{ locked}} \sum_{j \text{ locked}} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j) + Q_d \sum_{i \text{ locked}} \sum_{j \text{ floating}} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j) \\ & + Q_d \sum_{i \text{ floating}} \sum_{j \text{ locked}} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j) + Q_d \sum_{i \text{ floating}} \sum_{j \text{ floating}} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j) \\ \equiv {} & Z_{\mathrm{l,l}} + Z_{\mathrm{l,f}} + Z_{\mathrm{f,l}} + Z_{\mathrm{f,f}} \end{aligned}$$

Since we are minimizing and all $Z_{i,j}(\varkappa_i, \varkappa_j)$, w_i, and Q_d are positive, we can prune any search node which has $Z_{\mathrm{l,l}} >$ than the distance of the lowest distance configuration found thus far, since additional terms will only add to the sum.

We obtain a lower bound on the distance by lower-bounding each of the above four terms. Just taking the lowest value in the tables can yield differing values for any given variable, e.g., the values for a single variable $\varkappa_i$:

$$\varkappa_i^* = \arg\min_{\varkappa_i} Z_{i,j}(\varkappa_i, \varkappa_j), \quad \varkappa_i^* = \arg\min_{\varkappa_i} Z_{j,i}(\varkappa_j, \varkappa_i), \quad \text{and} \quad \varkappa_i^* = \arg\min_{\varkappa_i} Z_{i,i}(\varkappa_i)$$

may all be different. To correct this we write

$$Q_d Z_w = Z_{\mathrm{l,l}} + Z_{\mathrm{f,f}} + \sum_{i \text{ floating}} Z_{\mathrm{l,f}}^{(i)}(\varkappa_i^*)$$

where $Z_{\mathrm{l,l}}$ and $Z_{\mathrm{f,f}}$ are calculated as before and

$$Z_{\mathrm{l,f}}^{(i)}(\varkappa_i) = Q_d\Big(w_i^2 Z_{i,i}(\varkappa_i) + w_i \sum_{j \text{ locked}} w_j\big(Z_{i,j}(\varkappa_i, \varkappa_j) + Z_{j,i}(\varkappa_j, \varkappa_i)\big)\Big).$$

The best choice for the floating variable $\varkappa_i$ is

$$\varkappa_i^* = \arg\min_{\varkappa_i} Z_{\mathrm{l,f}}^{(i)}(\varkappa_i). \tag{34}$$

In this way all the possible choices for $\varkappa_i$ are at least consistent, and the true lower limit is still guaranteed to be higher than this (since the $Z_{\mathrm{f,f}}$ contribution can only be higher).

This approach also naturally suggests a variable ordering and a value ordering within any variable. We can try the variables in increasing order of their corresponding $Z_{\mathrm{l,f}}^{(i)}(\varkappa_i^*)$ so that variable i, which has the smallest $Z_{\mathrm{l,f}}^{(i)}(\varkappa_i^*)$, is tried first. For any given variable $\varkappa_i$ we can also order the possible value assignations for $\varkappa_i$ according to the array of values $Z_{\mathrm{l,f}}^{(i)}(\varkappa_i)$ so that the value $\varkappa_i = \varkappa_i^*$ is tried first and the value which maximizes $Z_{\mathrm{l,f}}^{(i)}(\varkappa_i)$ is tried last.

With little extra work we can impose a consistency requirement raising the lower bound for $Z_{\mathrm{f,f}}$. We decompose the sum as follows

$$Z_{\mathrm{f,f}} \equiv Q_d \sum_{i,j \text{ floating}: j \neq i} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j) = Q_d \sum_{i,j \text{ floating}: j < i} w_i w_j \big(Z_{i,j}(\varkappa_i, \varkappa_j) + Z_{j,i}(\varkappa_j, \varkappa_i)\big).$$

Instead of taking the minimum component-wise as previously suggested, *i.e* instead of

$$Z_{\mathrm{f,f}}^* = Q_d \sum_{i,j \text{ floating}: j \neq i} \min_{\varkappa_i, \varkappa_j} \big[w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j)\big],$$

we take the minimum so that

$$Z_{\mathrm{f,f}}^* = Q_d \sum_{i,j \text{ floating}: j < i} \min\big[w_i w_j \big(Z_{i,j}(\varkappa_i, \varkappa_j) + Z_{j,i}(\varkappa_j, \varkappa_i)\big)\big]. \tag{35}$$

which enforces pairwise consistency between variables i and j.

The lower bound, lb, we have obtained is thus

$$\text{lb} = Z_{\text{l,l}} + Z^*_{\text{f,f}} + \sum_i Z^{(i)}_{\text{l,f}}(\varkappa^*_i)$$

where $\varkappa^*_i$ is determined from Eq. (34) and $Z^*_{\text{f,f}}$ is determined from Eq. (35).

Integration Including Continuous and Range Contributions

In this section we summarize the complete heuristic specification, including the effects of the continuous and range variables. As above, we use the definitions

$$Z_{\text{l,l}} = Q_d \sum_{i,j \text{ locked}} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j)$$

$$Z^{(i)}_{\text{l,f}}(\varkappa_i) = Q_d w_i \sum_{j \text{ locked}} w_j \big(Z_{i,j}(\varkappa_i, \varkappa_j) + Z_{j,i}(\varkappa_j, \varkappa_i)\big)$$

$$Z^*_{\text{f,f}} = Q_d \sum_{i,j \text{ floating}: j<i} \min\big[w_i w_j \big(Z_{i,j}(\varkappa_i, \varkappa_j) + Z_{j,i}(\varkappa_i, \varkappa_j)\big)\big]$$

so that a lower bound to the complete discrete contribution to the distance is $Z_{\text{l,l}} + Z^*_{\text{f,f}} + \sum_i Z^*_{\text{l,f}}(\varkappa^*_i)$ where $\varkappa^*_i = \arg\min_{\varkappa_i} Z^{(i)}_{\text{l,f}}(\varkappa_i)$.

We now extend this result to include the continuous and range variables. The continuous and range variables depend on the discrete variables through the mask. Since we expect there will be relatively few masks we exhaustively enumerate all the discrete possibilities. Consequently, we define $d_1(\boldsymbol{\varkappa}_{\text{floating}}|\boldsymbol{\varkappa}_{\text{locked}})$ as the distance contribution made by the continuous and range variables (the d_1 contribution to Eq. (5)). We define the continuous + range contribution for any given floating variable i as

$$d^{(i)}_1(\varkappa_i) \equiv \min_{\text{floating}\setminus i} d_1(\boldsymbol{\varkappa}_{\text{floating}}|\boldsymbol{\varkappa}_{\text{locked}})$$

where the min is taken over all floating variables other than i and where floating variable i is locked to value $\varkappa_i$.

Since each continuous/range/discrete distance contributes additively to the total distance, we can rank the possible values for any given floating variable i according to $Z^{(i)}_{\text{l,f}}(\varkappa_i) + d^{(i)}_1(v_i)$. The best (according to this heuristic) setting for floating variable i is

$$\varkappa^*_i = \arg\min_{\varkappa_i} \Big\{ Z^{(i)}_{\text{l,f}}(\varkappa_i) + d^{(i)}_1(v_i) \Big\}.$$

The heuristic then suggests a variable ordering in order of increasing $Z^{(i)}_{\text{l,f}}(\varkappa^*_i) + d^{(i)}_1(v^*_i)$ so that we first select the floating variable with the lowest sum and last select the variable with the highest sum.

Finally, we turn to the lower bound on the distance. Noting that the continuous and range distance contribution does not have the form of a sum over the dimensions (as the discrete

contribution does), we write the lower bound, lb, as[34]

$$\mathrm{lb} = Z_{\mathrm{l,l}} + \sum_i \left(Z_{\mathrm{l,f}}^{(i)}(\varkappa_i^*) + \delta_{i,i^*} d_1^{(i)}(\varkappa_i^*) \right) + Z_{\mathrm{f,f}}^*$$

where $i^* = \arg\min_i \left(Z_{\mathrm{l,f}}^{(i)}(\varkappa_i^*) + d_1^{(i)}(v_i^*) \right)$ (i.e., i^* labels the floating dimension with the least $Z_{\mathrm{l,f}}^{(i)} + d_1^{(i)}$ contribution to the distance).

5 **E.2.2 Improvements**

We point out that an extensive literature exists on parallel branch and bound algorithms. Many of the techniques developed are applicable to the present problem.

PC-DOS 5.0

[34] $\delta_{x,x'}$ is the Kronecker delta function which is 1 if and only if $x = x'$.

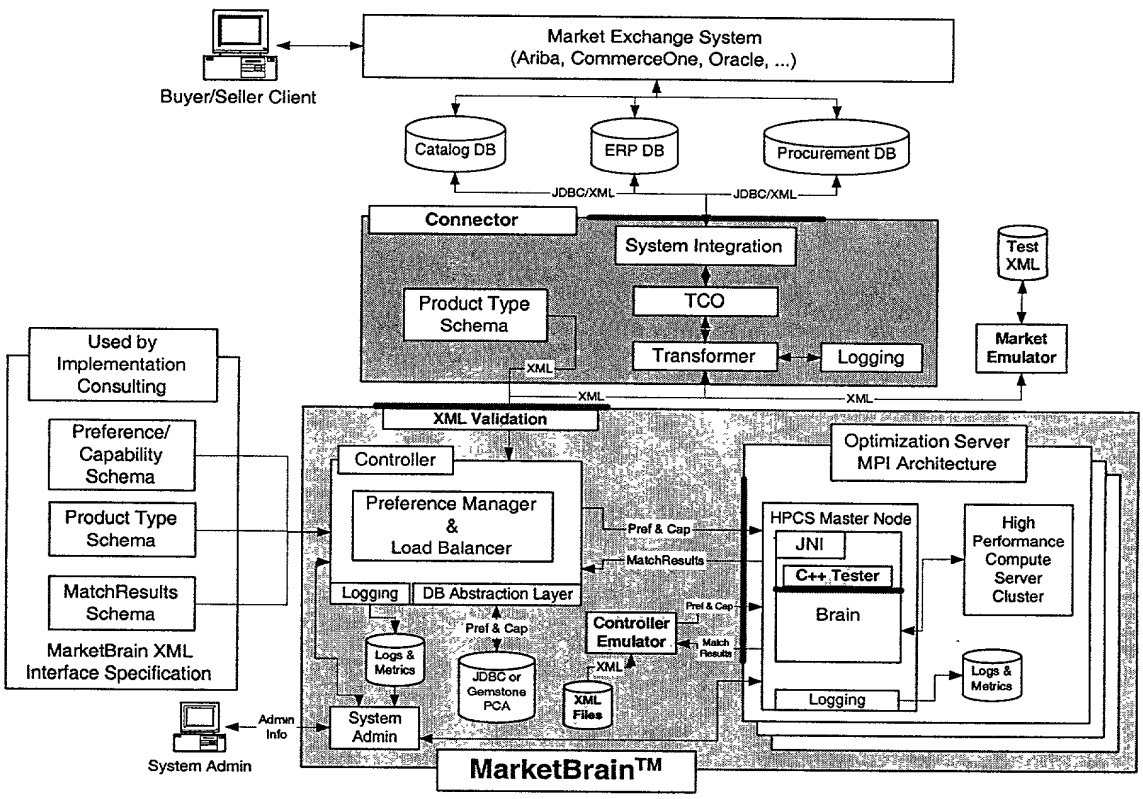


Figure 1: An architecture for the invention.

5,930,780

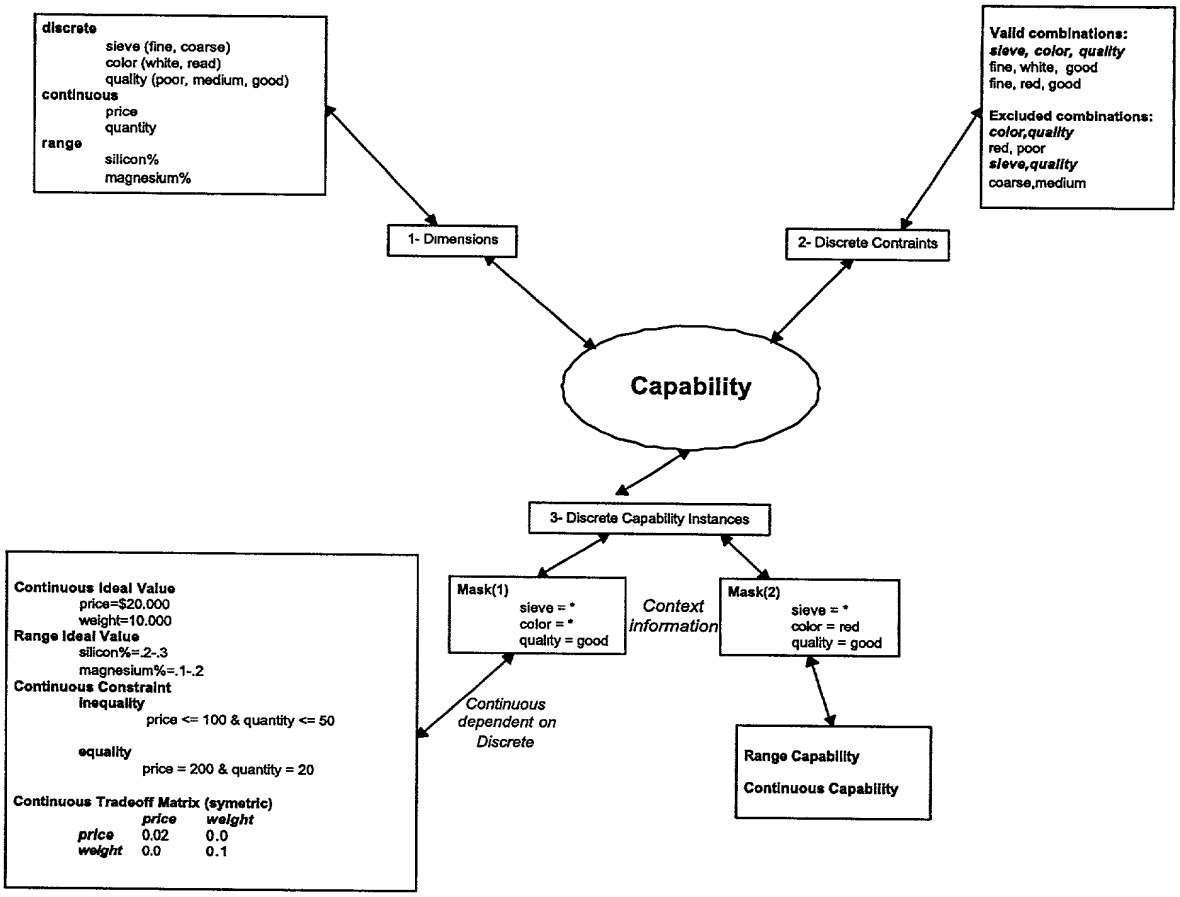


Figure 2: Schematic of a buyer-specific capability with examples indicating potential input.

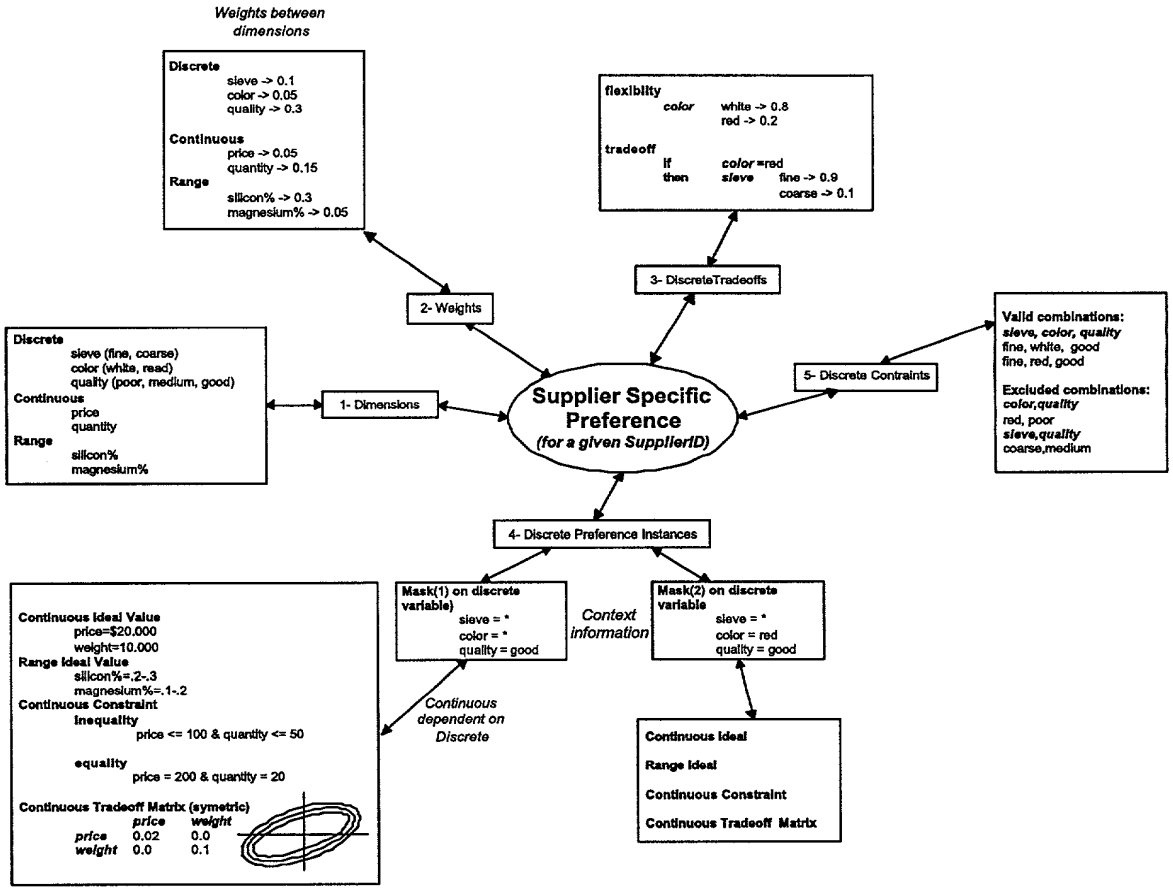


Figure 3: Schematic of a supplier-specific preference with examples indicating potential input.